UNIVERSIDAD DE MURCIA

Departamento de Ingeniería de la Información

y las Comunicaciones

Metaheurísticas híbridas cooperativas y su control usando Soft Computing y conocimiento previo

D. Enrique Muñoz Ballester

Murcia Junio 2010

Metaheurísticas híbridas cooperativas y su control usando Soft Computing y conocimiento previo

D. Enrique Muñoz Ballester

TESIS DOCTORAL

DIRECTORES:

Dr. D. José Manuel Cadenas Figueredo

Dra. Dª. María del Carmen Garrido Carrera

Junio 2010

DEPARTAMENTO DE INGENIERÍA DE LA INFORMACIÓN Y LAS COMUNICACIONES

FACULTAD DE INFORMÁTICA

UNIVERSIDAD DE MURCIA

Metaheurísticas híbridas cooperativas y su control usando Soft Computing y conocimiento previo

Memoria de Tesis presentada por

D. Enrique Muñoz Ballester

Directores

Dr. D. José Manuel Cadenas Figueredo

Dra. Dª. María del Carmen Garrido Carrera

Enrique Muñoz Ballester

José Manuel Cadenas Figueredo

María del Carmen Garrido Carrera

Junio 2010

DEPARTAMENTO DE INGENIERÍA DE LA INFORMACIÓN Y LAS COMUNICACIONES
UNIVERSIDAD DE MURCIA

UNIVERSIDAD DE MURCIA

D. Antonio Fernando Gómez Skarmeta, Catedrático de Universidad del Área de Ingeniería Telemática y presidente de la Comisìn Académica del Postgrado "Tecnologías de la Información y Telemática Avanzadas" de la Facultad de Informática de la Universidad de Murcia.

INFORMA:

Que la Tesis Doctoral titulada "Metaheurísticas híbridas cooperativas y su control usando Soft Computing y conocimiento previo", ha sido realizada por D. Enrique Muñoz Ballester, bajo la inmediata dirección y supervisión de D. José Manuel Cadenas Figueredo y Dª. María del Carmen Garrido Carrera, y que la Comisión ha dado su conformidad para que sea presentada ante la Comisión General de Doctorado.

D. Antonio F. Gómez Skarmeta

En Murcia, 18 de junio 2010

UNIVERSIDAD DE MURCIA

D. José Manuel Cadenas Figueredo y Dª. María del Carmen Garrido Carrera, Profesores Titulares de Universidad del Área de Ciencias de la Computación e Inteligencia Artificial en el Departamento de Ingeniería de la Información y las Comunicaciones de la Universidad de Murcia,

AUTORIZAN:

La presentación de la Tesis Doctoral titulada "Metaheurísticas híbridas cooperativas y su control usando Soft Computing y conocimiento previo", realizada por D. Enrique Muñoz Ballester, bajo nuestra inmediata dirección y supervisión, en el Departamento de Ingeniería de la Información y las Comunicaciones de la Universidad de Murcia, y que presenta para la obtención del grado de Doctor por la Universidad de Murcia.

José Manuel Cadenas Figueredo	María del Carmen Garrido Carrera
Director	Directora

En Murcia, 16 de junio de 2010

DEDICATORIA

A mis padres y a mi hermana

AGRADECIMIENTOS

En primer lugar a José Manuel y Mari Carmen auténticos artífices de esta tesis, sin cuyo apoyo, consejo y ánimo nunca habría sido posible convertir este proyecto en realidad. Y no sólo por su quehacer a nivel profesional, sino también a nivel personal y humano. Para mí ha sido una suerte y un privilegio tenerlos como directores de tesis.

A los miembros del Departamento de Ingeniería de la Información y las Comunicaciones de la Universidad de Murcia por los conocimientos, consejos y la ayuda dada en todo momento. En especial a mis compañeros de despacho, Andrés, Juanvi, Félix, José y Pascual, por el buen ambiente que han creado en el laboratorio y que ha hecho mucho más fácil los largos días en la facultad.

Al Ministerio de Ciencia e Innovación por el apoyo dado por medio de los proyectos TIN2005-08404-C04-02 y TIN2008-06872-C04-03. Así como a la Fundación Séneca, por el soporte dado para el desarrollo de este trabajo a través del programa de becas FPI. Dicha financiación también ha hecho posible mi estancia en Salerno (Italia), que también debo agradecer a los profesores Piero Bonissone y Vincenzo Loia. También quiero expresar mi agradecimiento a mis compañeros del LASA Lab y al resto de amigos italianos, en especial a Gianni y Giustino, por su hospitalidad y ayuda durante el tiempo que pasé en Italia.

A nivel personal, tampoco puedo olvidar a mis amigos Manu, Álex, Carlos, Tao y todos los demás, porque las risas, las cañas, las partidas, las fiestas, ..., también ayudan a hacer una tesis.

Y por supuesto a Mari, por quererme, aceptarme y soportarme, incluso en los peores momentos al final de esta tesis.

Por último, y con gran énfasis, agradecer a mi familia, mis padres, Isa y Quique, mi hermana Julia, y cómo no, a mis abuelas, Maruja y Carmen, por su cariño y apoyo incondicional y porque gran parte de lo que soy se lo debo a ellos.

SUMMARY AND CONCLUSIONS

This summary includes an English version of chapters 1 (introduction) and 13 (conclusions). It is incorporated here to meet the requirements for consideration as a PhD with European mention.

Introduction

Ever since humankind has been aware of itself it has to tackle the issue of solving the problems encountered in the environment. Many of these entail finding the best solution from among a number of valid ones available. Such optimization problems abound in our daily lives. All of us are continually solving optimization problems, like finding the shortest route from home to work in accordance with traffic constraints. We find efficient solutions to these problems because they are manageable. But there are optimization problems on much larger scales, such as how to make best use of a whole fleet of aircraft in terms of fuel saving, and to tackle these we need computational algorithms.

The search to solve these problems has given rise to a variety of strategies, and one of the most important of these is metaheuristics. Metaheuristics afford effective strategies to find approximate solutions to optimization problems. When working in this field, however, it is not uncommon to come up against the algorithm-instance problem or the problem of choice of algorithm, [232]. The question lies in determining which algorithm should be selected to solve a given instance, and by maximizing a performance measure. In any case, the "automatic algorithm selection problem is an undecidable one", [137], and the most common approach consists of measuring the performance of a set of algorithms and using the one that returns the best performance. However, we are of the opinion that a more interesting approach is to look for less rigid strategies that are able to adapt better to the changing conditions of the problems.

Hybrid systems offer flexible mechanisms for solving complex problems which are highly difficult to solve using less tolerant approaches, which means that a hybrid system is an interesting way of tackling the algorithm-instance problem, since an intelligent combination of different metaheuristics can be expected to afford more efficient and more flexible approaches. In particular, a cooperative metaheurístic, in which various metaheuristics work together, may be the most suitable strategy since greater robustness is supposed and higher quality solutions can be obtained for different classes of instances.

Nevertheless, combining different metaheuristics "intelligently" so that good results are obtained with all types of instances and problems supposes a tolerant approach, such as that afforded by "Soft Computing". Although the increased tolerance may lead to some loss of accuracy, this can be sacrificed in exchange for systems that are more robust to changes in problems and instances. Moreover, the use of knowledge that has been modelled with data mining techniques may help to adapt the behaviour of the system by configuring each of the metaheuristics and by helping in the decisions involving cooperation between them.

This work therefore proposes a general framework, which has been called KEPS, to design and build hybrid, parallel, adaptive and cooperative metaheuristics based on multi-agent systems. In a metaheuristic that has been designed with KEPS, various individual metaheuristics - some based on populations (Genetic Algorithms, Swarm Particle Optimization, Ant Colony, ...) and others on trajectories (Tabu Search, Simulated Annealing, Variable Neighborhood Search, ...) - cooperate to find the best solution possible for an instance of an optimization problem. In the execution of a KEPS type metaheuristic the parameters for each of the metaheuristics that comprise the same are chosen intelligently and are executed in *parallel* while *cooperating* by sharing their solutions in an *adaptive* way. Adaptivity is achieved through fuzzy rules that are able to assess the information extracted in the above process thanks to a previous, intelligent knowledge extraction process. The process studies earlier executions of the individual metaheuristics so as to learn how to adapt the cooperative metaheuristic to the different instances.

Given its parallel and distributed nature, the approach can be naturally modelled using a multi-agent system in which a set of *optimization agents* runs the cooperative metaheuristics under the supervision of a *coordinating agent* whose intelligence is supplied by the rules and the knowledge extraction process already outlined.

This field of work has been developed within the on-going HEURIMIND ("Extraction of useful knowledge by means of data mining for cooperative heuristics"[1]) project TIN2005-08404-C04-02, funded by the Spanish Ministry for Science and Innovation, along with research projects "Data mining based techniques for the Design of Intelligent Hybrid Systems"[2] (TIN2008-06872-C04-03) and "Analysis, study and development of Intelligent Systems and Telematic Services"[3] (04552/GERM/06) financed by the Spanish Ministry for Science and Innovation and the Regional Government of Murcia (Spain), respectively.

[1] In Spanish: "Extracción de conocimiento útil mediante minería de datos para heurísticas cooperativas"
[2] In Spanish: "Metodologías basadas en minería de datos para el diseño de Sistemas Híbridos Inteligentes"
[3] In Spanish: "Análisis, estudio y desarrollo de Sistemas Inteligentes y Servicios Telemáticos"

The specific objectives of this thesis are:

- To deepen understanding in the field of cooperative strategies and in the current applications of Soft Computing techniques.
- The design and specification of a data generation protocol for the study of the algorithm-instance relation (heuristics-problems in our case).
- The computational checking of the data generation protocol in Soft Computing environments.
- The deeper study of intelligent knowledge extraction techniques (Data Mining) and of the various ways of tackling the different stages of the process so as to be able to apply them to heuristics-problems.
- The application of different Data Mining techniques to the information generated by heuristics-problems.
- The extraction of useful knowledge both by more efficient problem solving and by improvements to cooperative metaheuristics that have bestowed adaptivity.
- The proposal of a general framework for the design and construction of hybrid, parallel, adaptive and cooperative metaheuristics based on multi-agent systems.
- The assessment of various cooperative metaheuristics within the general framework applied to both laboratory problems and to real ones.

To achieve these aims the Thesis is divided into 13 Chapters (including the Introduction and the Conclusions), grouped in 4 Parts that describe the work carried out.

Chapters 2 and 3 (Part I) include a review of the state-of-the-art of optimization problems and in the areas of metaheuristics and cooperative metaheuristics. Specifically, by studying the various taxonomies [89,161,226] that take in cooperative metaheuristics it has been shown that in spite of the large number of proposals, there are many areas that remain to be explored. In particular, it is observed that there has been little study of heterogeneous cooperative metaheuristics, yet these are considered as the most promising. Within these, it has been shown that there are no existing approaches to tackle problems by using previous knowledge about them, so it could be argued that they are acting blind. It is on this type of strategies that the work presented here centres. It defines a general framework for their design and construction.

Part II introduces the KEPS general framework for the design and construction of cooperative metaheuristics using previous knowledge and Soft Computing. Initially, Chapter 4 introduces basic concepts that will lead to a better understanding of the proposal. These include multi-agent systems, on which the architecture of the proposed framework is based, logic and fuzzy control systems, on which the control strategy, the intelligent knowledge extraction and the data mining processes used for extracting previous knowledge are based. Chapter 5 develops the general framework with the definition of its architecture, which is modelled using a multi-agent system in which several optimization agents cooperate under the supervision of a coordinating agent. Their cooperation schema is controlled by a fuzzy control system and is able to use modelled knowledge by means of data mining techniques. Chapter 6 analyses the impact of the different parameters that make up the general framework. Of special mention among these are the composition of the metaheuristic generated, the architecture used, the cooperation schema and the number of exchanges permitted. This Part of the thesis is concluded in Chapter 7 where a similar study is performed but, unlike the previous one, which was governed by the number of evaluations of the objective function, here the execution is controlled by time.

Following the application of the general framework to several problems, so generating an equivalent number of KEPS type metaheuristics, it was observed that intelligent knowledge extraction process used in the design might require a number of computational resources. Thus, in Chapter 11 of Part IV, an evolution of KEPS is studied in order to improve the knowledge extraction process. This uses techniques which can reduce the number of resources while not affecting the knowledge models obtained.

Finally, the various proposals need to be validated, so in Part III and in Chapter 12 of Part IV, a series of applications were made to problems in different fields, such as e-Learning (Chapter 8), automatic music composition (Chapter 9), localization problems (Chapter 10) and dynamic problems (Chapter 12).

Figure 1 shows a diagram that synthesizes the project presented in this thesis.

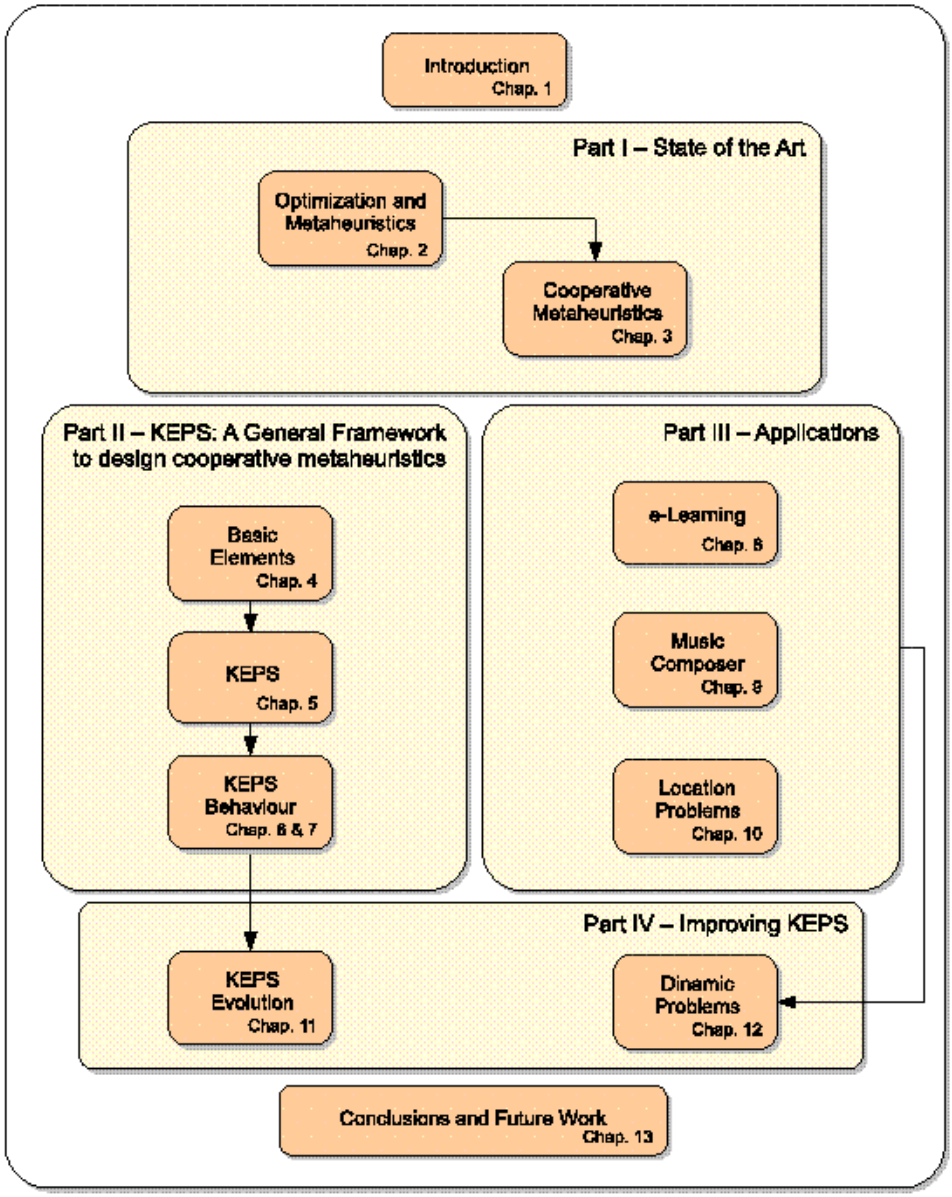


Figura 1: General diagram of the dissertation

Conclusions

A general framework for the design and construction of hybrid, parallel, adaptive and cooperative metaheuristics based on multi-agent systems is presented under the name of KEPS. Several metaheuristics, based on populations and trajectories alike, cooperate to find the best solution possible for an instance of an optimization problem. Specifically, a KEPS type metaheuristic intelligently chooses the parameters of each of the individual metaheuristics it is made up according to the instance to be solved. These are then executed in *parallel* and they *cooperate* by sharing their solutions in an *adaptive* way. Adaptivity is achieved through the use of a set of fuzzy rules that can assess the information extracted in preliminary knowledge extraction process. The proposal has been applied to a range of optimization problems of various types (combinatory and continuous, static and dynamic, real world and laboratory) and satisfactory results have been obtained that demonstrate its robustness and adaptivity.

Below we list the advances that have been min terms of the aims established at the outset (Chapter 1):

1. *To deepen the knowledge of the field of cooperative strategies and the current applications of Soft Computing.*

   A wide study of cooperative strategies has been carried out with the focus on those that have attained a certain degree adaptability through the use of Soft Computing techniques, like cooperative metaheuristics. A lack of strategies that make use of previous knowledge to adapt to the various problems and instances has been observed. Of particular interest is the inclusion of this previous knowledge through Soft Computing techniques. Fuzzy control and fuzzy learning models are valid techniques for this purpose.

   With this aim, cooperative metaheuristics have been studied along with the impact that the use of previous knowledge incorporated through Soft Computing techniques has on them. During the course of this study and some preliminary tests, improvements have been observed when cooperative strategies that use previous knowledge are employed.

2. *Design and Specification of a data generation protocol to study the algorithm-instance protocol (heuristics-problems in this case).*

   Solving problems with cooperative metaheuristics that use previous knowledge has been shown to be efficient. However, to obtain this knowledge it is necessary define an intelligent knowledge extraction process that gives the configuration of the coordinator from the earlier executions of the metaheuristics. For each instance of the problem, this process is in responsible for: 1) ascertaining which parameter values are best for

each metaheuristics; 2) obtaining a weight-based ranking from among the cooperating metaheuristics. In other words, for carrying out an automatic study of the algorithm-instance relation.

Thus, in Chapter 5 there is a definition of an intelligent knowledge extraction process that aids in collecting algorithm-instance relations and, even if it is not able to solve the algorithm-selection problem, it does use previous knowledge to discern how to configure a cooperative metaheuristic so that it can automatically adapt to the instance to be solved. Within this extraction process, the first phase defined corresponds to the protocol mentioned earlier. The information generated by the execution of various metaheuristics is gathered and structured so that it can be exploited. Specifically, a protocol is defined that is divided into two parts. In the first the metaheuristics are executed several times using different combinations of parameters and storing the information generated in "raw" data sets. In the second part this information is refined and data sets known as "refined" are obtained. The various steps in the design of the data generation protocol have been published at congresses [49, 52, 53, 61, 62], in the chapter of a book [63] and in scientific journals [55] and [2] (submitted) A tool has also been proposed that supports this phase and the advances made have been published at congresses [45, 60] and in a book [46].

3. *Checking the data generation protocol computationally.*

   The protocol has been used in various problems and from the results it can be concluded that the data generated contain useful information about the algorithm-instance relation. However, its application supposes the use of many computational resources, which can lead to longer application times. Hence, in, Chapter 11, a modification of the protocol is considered in order to reduce resource consumption.

4. *To study knowledge extraction (Data Mining) in greater depth along with the diffferent ways of tackling the stages involved in these techniques so as to be able to apply them to heuristics - problems.*

   A knowledge extraction process was defined to study the algorithm-instance relation. A preliminary study was carried out of the different stages of the process (see Appendix C). This appendix also includes a review of the data mining techniques considered to be of most interest for the process defined. On the basis of this study, it was decided that one of the most suitable techniques to meet our aims was fuzzy decision trees, on account of their power and comprehensiveness. The appropriateness of this choice has been borne out by the results obtained on applying the KEPS general framework to

solve different problems. The various steps followed to define the knowledge extraction process have been published at congresses [49,52,53,61,62], in the chapter of a book [63] and in scientific journals [55] and [2] (submitted). Furthermore, a fuzzification process of the data submitted at congresses was defined in order to help in obtaining the fuzzy tree models [50].

The problem of resource consumption caused by the data generation protocol led us to define a new knowledge extraction process that used techniques that were able to alleviate the problem. It was observed that active learning techniques would serve to significantly reduce the use of resources without affecting the models generated. A new process was therefore designed based on active learning so as to obtain the algorithm-instance relation. However, the fuzzy decision trees showed that they were not able to adapt to the new requirements of the process. Taking as our base the study shown in Appendix C, we used the SVM technique as a possible new option. Its effectiveness was proved when it was applied to the tests carried out in Chapters 11 and 12. The new, active learning based, process was published at congress [4].

5. *Application of the different Data Mining techniques to information generated by heuristics - problems.*

   Both knowledge extraction processes and, therefore, the different data mining techniques proposed were applied to problems of various natures and satisfactory results were obtained in all cases. It was therefore shown that the techniques chosen are able to extract useful data for solving optimization problems.

6. *Extraction of useful knowledge to solve problems efficiently and to make improvements in cooperative metaheuristics.*

   As commented above, the knowledge extraction process has been applied to a range of problems. In all cases, the cooperative metaheuristics that were configured through the extraction process improved their results with respect to those that use the process, so we can state that the knowledge extracted was indeed useful and that it helps in the execution of cooperative metaheuristics.

7. *To propose a general framework for the design and construction of hybrid, parallel, adaptive and cooperative metaheuristics based on multi-agent systems.*

   At this stage, a general framework is developed for the design of cooperative metaheuristics that envelops the design of a multi-agent architecture in which all types of metaheuristics are able to cooperate under the supervision of a coordinator that uses previously,

intelligently-extracted information to choose the parameters of each of the metaheuristics intelligently and to control the sharing of solutions. The use of acquired knowledge has been made through a set of fuzzy rules which decide when and how the various metaheuristics should share information. The definition of the framework was published in a scientific article [55].

Once the general framework has been defined, efforts were directed towards going more deeply into the study of the impact of the parameters that affect its behaviour. Chapter 6 considers the influence of the composition of the cooperative metaheuristic and compares several homogeneous compositions (i.e., made up of a single individual replicated metaheuristic individual) with a heterogeneous one (formed by a number of individual metaheuristics) and different individual metaheuristics. From these comparisons it is deduced that a homogeneous cooperative metaheuristic always outperforms its individual counterpart and how it is the heterogeneous metaheuristic that returns the best results. What type of architecture is the most suitable is then studied: 1)a two-tier architecture, with two sequential phases - one to do with population based metaheuristics and another with trajectory based ones; or 2) a single-tier architecture in which all the metaheuristics are run in parallel. The results show that both architectures are valid, as they both return similar results. The next step was to check which cooperation schema produced the best results - one based on the memory in the execution or on previously acquired knowledge. The results show a noticeable improvement when using the second approach, and the conclusion is that the use of a previous learning process is of help to the cooperative metaheuristics. The effect of the number of solution exchanges between the individual metaheuristics is checked. The results show that too high a number can lead to bad results, while too few exchanges also influence negatively. An intermediate number of exchanges gives very satisfactory results. Chapter 7 likewise studies the benefits of cooperation and of the use of knowledge when the execution time is one of the variables governing the cooperation. Finally, Chapter 11 examines the effects of the data mining model used in the intelligent knowledge extraction process. The study of the impact of the parameters influencing the behaviour of the metaheuristics generated under this framework has been the source for the publication of chapter in books [54,58].

8. *To assess the various cooperative metaheuristics within the general framework when applied to laboratory and real problems.*

   The different cooperative metaheuristics built have been applied to various problems. In Chapters 6, 7, 8, 9, 10 and 12 there are tests with combinatory problems, while the continuous problem is tested out in Chapter 12. Elsewhere, in Chapters 6, 7, 8, 9

and 10, tests are made with static problems, while Chapter 12 deals with two dynamic problems. Moreover, Chapters 6, 7, 10 and 12 study laboratory problems, as opposed to Chapters 8 and 9 which look at real world problems. Finally, a localization problem was undertaken in el Chapter 10. It should be highlighted that very good results were obtained for all the problems tested. The various applications of the general framework proposed have led to the publication of congress communications [6, 56, 57, 59], a book chapter [64], as well as articles in scientific journals [2, 3].

PRÓLOGO

Esta tesis presenta una propuesta relacionada con la resolución de problemas de optimización utilizando metaheurísticas cooperativas. La propuesta abarca el diseño, desarrollo y aplicación de estas técnicas a diversos ámbitos dentro de los problemas de optimización. El objetivo es la obtención de un marco general para el diseño y construcción de metaheurísticas cooperativas. Las metaheurísticas construidas utilizando este marco deberán ser robustas y capaces de adaptarse a los distintos problemas de optimización que surgen en un entorno real, así como a las distintas instancias que puedan aparecer en cada problema.

Para ello se propone una arquitectura multiagente en la que metaheurísticas de diversa índole (tanto basadas en trayectorias, como en poblaciones) cooperan bajo la supervisión de un coordinador que, haciendo uso de técnicas de soft computing y modelos de conocimiento obtenidos utilizando técnicas de minería de datos, es capaz de adaptarse a los distintos problemas de optimización e instancias que puedan aparecer, obteniendo resultados competitivos.

Se ha estudiado la influencia de los parámetros que configuran la metaheurística: metaheurísticas que la componen, tipo de estructura, esquema de cooperación, frecuencia de intercambios, proceso de extracción inteligente de conocimiento. En base a estos estudios se ha comprendido el funcionamiento de la estrategia y se han propuesto mejoras para paliar sus puntos débiles.

Por último, se han resuelto diversos problemas de optimización de toda índole: problemas de optimización combinatoria y continua, problemas estáticos y dinámicos, problemas del mundo real y de laboratorio... Los resultados obtenidos en todos los casos han sido satisfactorios.

En la figura 2 se observa un esquema que sintetiza el proyecto presentado en esta tesis.

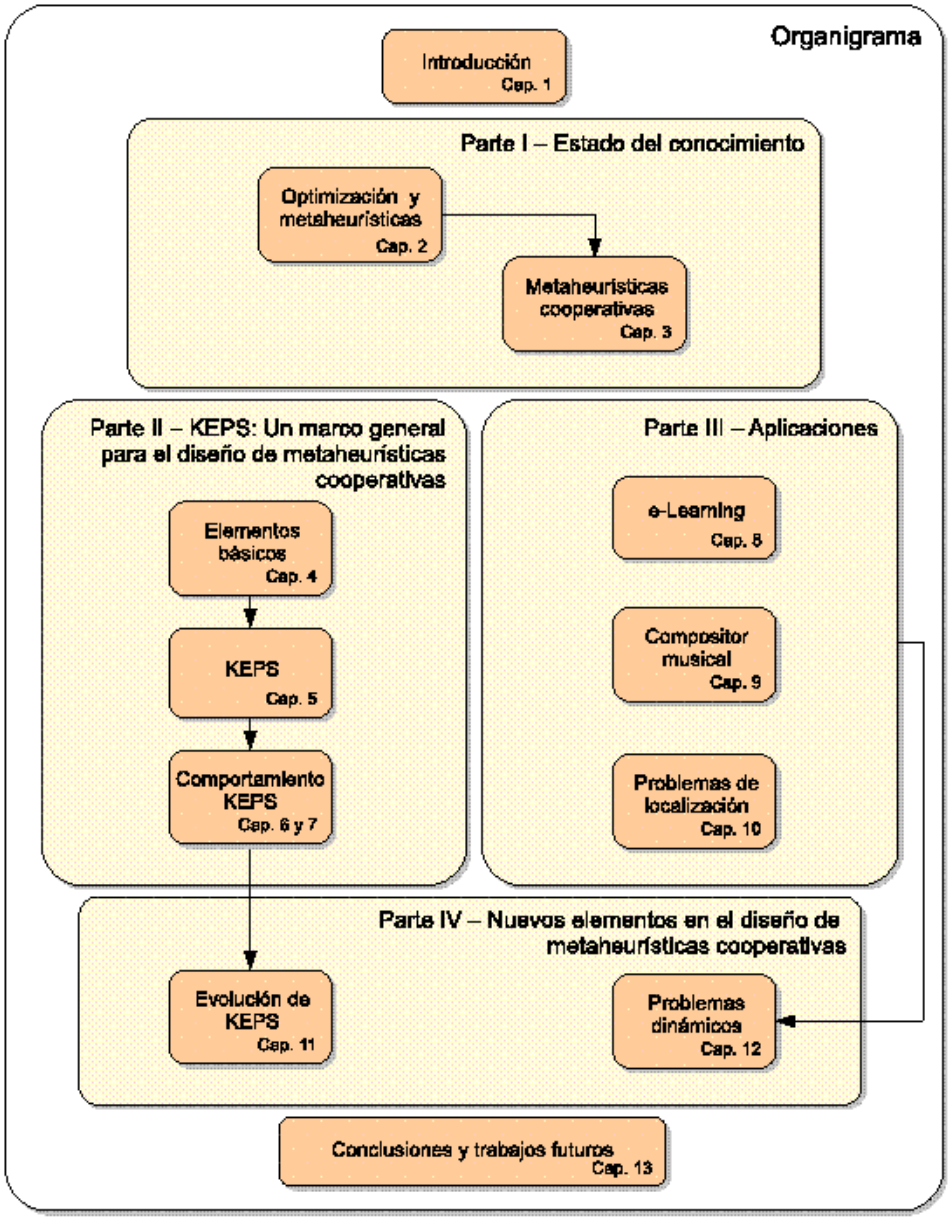


Figura 2: Esquema general de la memoria

Índice general

Capítulo 1

Introducción

Desde que el ser humano ha tenido conciencia de sí mismo, ha debido enfrentarse a la tarea de resolver los problemas que su entorno le ha presentado. Muchos de estos problemas consisten en obtener la mejor solución de entre el conjunto de soluciones válidas posibles. A este tipo de problemas se les conoce por el nombre de problemas de optimización y son muy comunes en la vida diaria. Cada persona constantemente resuelve problemas de optimización tales como encontrar el camino más corto desde su casa al trabajo sujeto a las restricciones del tráfico. Los humanos encontramos soluciones a ellos de una forma eficiente siendo esto posible porque son tratables. Sin embargo, los problemas de optimización pueden crecer a grandes escalas, como por ejemplo hacer más beneficioso el uso de una flota de aviones para reducir el costo de combustible. Estos tipos de problemas requieren de algoritmos computacionales para abordarlos.

Por ello para resolver estos problemas surgen diversas estrategias, de las cuales unas de las más destacadas son las metaheurísticas. Las metaheurísticas brindan estrategias efectivas para encontrar soluciones aproximadas a problemas de optimización. Al trabajar con ellas, sin embargo, no es extraño encontrarnos con el problema algoritmo-instancia o problema de la selección del algoritmo, [232]. Este problema consiste en determinar qué algoritmo se debe seleccionar para resolver una instancia dada, maximizando una medida de rendimiento. En cualquier aso el "problema de la selección automática del algoritmo es indecidible", [137], y el enfoque más común para resolverlo consiste en medir el rendimiento de un conjunto de algoritmos sobre un conjunto de instancias, y utilizar aquél que presente el mejor. Sin embargo, nosotros consideramos más interesante buscar estrategias menos rígidas que permitan adaptarse mejor a las condiciones cambiantes de los problemas.

Los sistemas híbridos proporcionan mecanismos flexibles que resuelven problemas complejos, muy difíciles de resolver con otros enfoques menos tolerantes. Por tanto, un sistema híbrido es una forma interesante de atacar el problema algoritmo-instancia, puesto que es de

esperar que una combinación inteligente de diferentes metaheurísticas proporcione enfoques más eficientes y mayor flexibilidad. En particular, una metaheurística cooperativa, en la que las distintas metaheurísticas cooperen puede ser la estrategia más adecuada puesto que de esta manera se puede esperar una mayor robustez por el uso de diferentes metaheurísticas alcanzando soluciones de muy alta calidad para diferentes clases de instancias.

Pero combinar diferentes metaheurísticas de manera "inteligente" de tal manera que se obtengan buenos resultados con todo tipo de instancias y problemas requiere un enfoque tolerante, como el que proporciona el "Soft Computing". Aunque el aumento de tolerancia pueda provocar cierta pérdida de precisión, ésta se puede sacrificar para obtener sistemas más robustos ante los cambios de problemas e instancias. Por otro lado la utilización de conocimiento previo modelado utilizando técnicas de minería de datos puede ayudar a adaptar el comportamiento del sistema configurando cada una de las metaheurísticas y asistiendo en las decisiones que impliquen la cooperación entre ellas.

Por tanto, en este trabajo se propone un marco general para el diseño y construcción de metaheurísticas híbridas, paralelas, adaptativas y cooperativas basadas en sistemas multiagente, al que se ha denominado KEPS. En una metaheurística diseñada usando KEPS, diversas metaheurísticas individuales, tanto basadas en poblaciones (Algoritmos Genéticos, Optimización por Enjambre de Partículas, Colonia de Hormigas...) como basadas en trayectorias (Búsqueda Tabú, Temple Simulado, Búsqueda por Entornos Variables...), cooperan para encontrar la mejor solución posible a una instancia de un problema de optimización. Durante la ejecución de una metaheurísticas tipo KEPS, ésta elige de manera inteligente los parámetros de cada una de las metaheurísticas que la componen, las cuales se ejecutan *paralelamente* mientras *cooperan* intercambiando sus soluciones de un modo *adaptativo*. La adaptatividad se consigue mediante el uso de un conjunto de reglas fuzzy capaz de evaluar la información extraída por un proceso previo de extracción inteligente de conocimiento. Dicho proceso estudia ejecuciones previas de las metaheurísticas individuales para aprender como adaptar la metaheurística cooperativa a las distintas instancias.

Debido a su naturaleza paralela y distribuida, la aproximación puede modelarse de un modo natural utilizando un sistema multiagente, donde un conjunto de *agentes de optimización* ejecutan las metaheurísticas cooperativas bajo la supervisión de un *agente coordinador*, cuya inteligencia es proporcionada por las reglas y el proceso de extracción del conocimiento antes mencionados.

Esta área de trabajo se ha desarrollado en el seno del proyecto HEURI-MIND ("Extracción de conocimiento útil mediante minería de datos para heurísticas cooperativas" - TIN2005-08404-C04-02) financiado por el Ministerio de Ciencia e Innovación (España) junto con los

proyectos de investigación "Metodologías basadas en minería de datos para el diseño de Sistemas Híbridos Inteligentes" (TIN2008-06872-C04-03) y "Análisis, estudio y desarrollo de Sistemas Inteligentes y Servicios Telemáticos" (04552/GERM/06) financiados por el Ministerio de Ciencia e Innovación (España)) y la comunidad autónoma de Murcia (España) respectivamente y actualmente activos.

En este contexto, los objetivos concretos de esta tesis son los siguientes:

1. Profundizar en el campo de las estrategias cooperativas y la aplicación actual de técnicas de Soft Computing.
2. Diseñar y especificar un protocolo de generación de datos para el estudio de la relación algoritmo-instancia (en nuestro caso, heurísticas - problemas).
3. Comprobar computacionalmente el protocolo de generación de datos en ambientes propios de la Soft Computing.
4. Profundizar en el estudio de técnicas de extracción inteligente de conocimiento (Data Mining) así como en las distintas formas de abordar las fases que conllevan este proceso con el objetivo de aplicarlas en el conjunto heurísticas - problemas.
5. Aplicar las distintas técnicas de Minería de Datos a la información generada por el conjunto heurísticas - problemas.
6. Extraer conocimiento útil tanto para la resolución de problemas de manera más eficiente como para realizar mejoras en las metaheurísticas cooperativas proporcionándoles adaptatividad.
7. Proponer un marco general para el diseño y construcción de metaheurísticas híbridas, paralelas, adaptativas y cooperativas basadas en sistemas multiagentes.
8. Evaluar distintas metaheurísticas cooperativas dentro del marco general aplicadas a problemas tanto de laboratorio como reales.

Para el desarrollo de estos objetivos, esta memoria de tesis se ha estructurado en 11 Capítulos agrupados en 4 Partes que reflejan las tareas realizadas.

En los Capítulos 2 y 3 (Parte I) se realiza un estudio del estado del conocimiento en los problemas de optimización, y en las áreas de las metaheurísticas y las metaheurísticas cooperativas. En concreto, estudiando las distintas taxonomías [89, 161, 226] que engloban a las metaheurísticas cooperativas se ha constatado que, a pesar de la cantidad de propuestas, todavía quedan muchos campos por explorar. En particular, se observa un escaso estudio

de las metaheurísticas cooperativas heterogéneas, que sin embargo son consideradas las más prometedoras. Dentro de éstas, se ha comprobado que no existen aproximaciones que sean capaces de abordar los problemas utilizando conocimiento previo acerca de los mismos, por lo que se podría decir que actúan a ciegas. Es en este último tipo de estrategias en las que se centra el trabajo aquí expuesto, definiendo un marco general para su diseño y construcción.

En la Parte II, se introduce el marco general KEPS para el diseño y construcción de metaheurísticas cooperativas usando conocimiento previo y Soft Computing. Inicialmente, en el Capítulo 4 se introducen conceptos básicos que permitirán entender mejor la propuesta. Estos conceptos incluyen los sistemas multiagentes, en los que se basa la arquitectura el marco propuesto, la lógica y los sistemas de control fuzzy, en los que se basa la estrategia de control y los procesos de extracción inteligente de conocimiento y la minería de datos, utilizados para extraer el conocimiento previo. A continuación, en el Capítulo 5, se desarrolla el marco general definiendo su arquitectura, modelada utilizando un sistema multiagente en el que varios agentes de optimización cooperan bajo la supervisión de un agente coordinador, y su esquema de cooperación, controlado mediante un sistema de control fuzzy capaz de utilizar el conocimiento modelado mediante técnicas de minería de datos. Después, en el capítulo 6, se analiza el impacto de los distintos parámetros que configuran el marco general. Entre ellos cabe destacar: la composición de la metaheurística generada, la arquitectura utilizada, el esquema de cooperación y el número de intercambios permitidos. Para concluir esta parte, en el Capítulo 7 se realiza un estudio similar, pero mientras que en el anterior la ejecución de la metaheurística estaba gobernada por el número de evaluaciones de la función objetivo, en este la ejecución se controla utilizando el tiempo.

Después de aplicar el marco general a diversos problemas generando un número equivalente de metaheurísticas de tipo KEPS, se observó que el proceso de extracción inteligente de conocimiento utilizado en su diseño podía requerir la utilización de muchos recursos computacionales. Por ello, en el capítulo 11 de la Parte IV, se estudia una evolución de KEPS para mejorar el proceso de extracción de conocimiento que utiliza técnicas capaces de reducir la utilización de recursos sin afectar a los modelos de conocimiento obtenidos.

Por último, para validar las distintas propuestas, en la Parte III y en el Capítulo 12 de la Parte IV, se realizan distintas aplicaciones a problemas de diversos campos, como son: el e-learning (Capítulo 8), la composición automática de música (Capítulo 9), los problemas de localización (Capítulo 10) y los problemas dinámicos (Capítulo 12).

Parte I

ESTADO DEL CONOCIMIENTO

Capítulo 2

Optimización y Metaheurísticas

2.1. Problemas de Optimización

En general los ordenadores son usados para ejecutar algoritmos para la resolución de problemas. De hecho, estos problemas tienen características muy variables. Un problema puede ser expresado [14] en términos de una relación $P \subseteq I \times S$, donde I es el conjunto de *instancias del problema* y S es un conjunto de *soluciones del problema*. Si queremos analizar las propiedades de las computaciones que se deben realizar, es necesario considerar las características de los conjuntos I, S y de la relación P más en detalle.

En algunos casos, sólo se quiere determinar si una instancia $x \in I$ satisface una condición dada, es decir, si $\pi(x)$ se verifica, donde π es un predicado (unitario) especificado. Esto ocurre, por ejemplo, cuando queremos probar si un programa es sintácticamente correcto o un cierto número es primo. En estos casos, la relación P se reduce a una función $f : I \to S$, donde S es el conjunto binario $S = \{SI, NO\}$ (o $S = \{0, 1\}$), y se denota el problema como un problema de *decisión* (o reconocimiento). También podemos considerar problemas de *búsqueda*, donde, para cualquier instancia $x \in I$, se tiene que devolver una solución $y \in S$ que cumpla $(x, y) \in P$. Esto incluye aquellos problemas como, por ejemplo, encontrar un camino en un grafo entre dos nodos dados o determinar la (única) factorización de un entero. En otros casos, dada una instancia $x \in I$, se está interesado en encontrar la "mejor" solución y^* (de acuerdo con alguna medida) de entre todas las soluciones $y \in S$ tales que $(x, y) \in P$ se cumple. Este es el caso cuando, dado un punto q y un conjunto de puntos Q en el plano, se quiere determinar el punto $q' \in Q$ que es el más cercano a q o cuando, dado un grafo con pesos, se quiere encontrar un ciclo Hamiltoniano, si existe, de costo mínimo. Los problemas de este tipo se llaman *problemas de optimización* y ocurren con frecuencia en la mayoría de actividades humanas desde el comienzo de la historia de las matemáticas.

Un problema de optimización P se caracteriza [14] por la siguiente cuádrupla de objetos $(I_P, SOL_P, m_P, goal_P)$, donde:

1. I_P es el conjunto de instancias de P.
2. SOL_P es una función que asocia a cada instancia de entrada $x \in I_P$ el conjunto de soluciones factibles de x.
3. m_P es la función de medida, definida por pares (x, y) tal que $x \in I_P$ e $y \in SOL_P(x)$. Por cada par (x, y), $m_P(x, y)$ proporciona un valor de la solución factible y.
4. $goal_P \in \{MIN, MAX\}$ especifica si P es un problema de maximización o minimización.

Dada una instancia de entrada x, se denota como $SOL_P^*(x)$ al conjunto de *soluciones óptimas* de x, que es el conjunto de soluciones cuyo valor es óptimo (mínimo o máximo dependiendo de $goal_P$). Mas formalmente, para cada $y^*(x)$ tal que $y^*(x) \in SOL_P^*(x)$:

$$m_P(x, Y^*(x)) = goal_P\{m_P(x, z) \ / \ z \in SOL_P(x)\}$$

El valor de cualquier solución óptima $Y^*(x)$ de x se denota como $m_P^*(x)$.

Los problemas de optimización se suceden en nuestra vida diaria en cualquier lugar [32]. Cada uno de nosotros constantemente resolvemos problemas de optimización tales como encontrar el camino más corto desde nuestra casa al trabajo sujeto a las restricciones del tráfico u organizar nuestra agenda. Y los humanos encontramos soluciones a estos problemas diarios de una forma eficiente. ¿Por qué somos capaces de hacerlo? Porque son problemas todavía tratables, tienen una dimensión lo suficientemente pequeña como para procesarlos. Sin embargo, estos problemas pueden crecer a grandes escalas, tal como por ejemplo hacer más beneficioso el uso de una flota de aviones para reducir el costo de combustible. Entonces estos tipos de problemas son de tan alta dimensión y complejidad que se requieren de algoritmos computacionales para abordarlos.

Para modelar estos problemas, lo primero es identificar las posibles decisiones que pueden tomarse y esto puede hacerse a través de las variables del problema concreto que son generalmente de carácter cuantitativo, buscándose los valores de las mismas que optimizan el objetivo. Posteriormente se define qué decisiones resultan admisibles lo que conduce a un conjunto de restricciones teniendo presente la naturaleza del problema. Se calcula el coste/beneficio asociado a cada decisión admisible lo que supone determinar una función objetivo que asigna un valor de coste/beneficio a cada conjunto posible de valores para las variables que conforman una decisión.

La resolución de este tipo de problemas tuvo su auge a partir de la Segunda Guerra Mundial. Todo se inicia en Gran Bretaña en 1938 cuando a instancias del Ministerio del Aire

un equipo multidisciplinario de científicos comienza a investigar como integrar la información tomada de los radares con la de observadores de tierra para interceptar los bombarderos alemanes. En la década de los años cincuenta con la aparición de los computadores paralelos se comienza a aplicar en gran escala esta ciencia en el mundo empresarial como herramienta para la planificación y administración de negocios.

Existen problemas de optimización para los cuales no se conocen algoritmos de resolución, debido a que se produce una explosión exponencial del tiempo de calculo al aumentar su tamaño. Son problemas computacionalmente difíciles de tratar. Por el contrario para otros problemas de optimización existen algoritmos que solo crecen de forma polinomial en el tiempo con el tamaño del problema.

Matemáticamente se han caracterizado a ambos. Aquellos, para los cuales se conocen algoritmos que necesitan un tiempo polinomial para ofrecer la solución optima, se dice que pertenecen a la clase P, y se considera que son resolubles eficientemente. Sin embargo, se comprobó que la mayoría de los principales problemas de optimización pertenecen a un superconjunto de P, la clase denominada NP, en la cual se incluyen aquellos problemas para los que no se conoce un algoritmo polinomial de resolución.

La importancia de esta clase de problemas de decisión es que contiene muchos problemas de búsqueda y de optimización para los que se desea saber si existe una cierta solución o si existe una mejor solución que las conocidas. En esta clase están el problema del viajante (también llamado "problema del agente de ventas" o "problema del agente viajero") donde se quiere saber si existe una ruta óptima que pasa por todos los nodos en un cierto grafo y el problema de satisfacibilidad booleana en donde se desea saber si una cierta fórmula de lógica proposicional puede ser cierta para algún conjunto de valores booleanos para las variables.

Estos problemas difíciles tienen la peculiaridad de que si se pudiera obtener una solución en tiempo polinomial para uno de ellos, se podría lograr para todos los de la clase NP. Esta propiedad ha sido usada para definir una subclase separada en NP: la de los problemas NP-hard. Se dice que un problema es NP-hard si cualquier problema en NP es polinomialmente transformable en él, aunque el problema en sí no pertenezca a NP. Si el problema además pertenece a NP, entonces se le denomina NP-completo.

Todos los intentos de probar teóricamente que "P=NP" han fracasado. Aun más, puesto que no se ha encontrado ningún algoritmo exacto de tiempo polinomial para ningún problema NP, a pesar de muchos años y esfuerzos, hay una fuerte evidencia circunstancial de que "P $\neq$ NP", que habla en favor de encontrar formas alternativas de "resolver" problemas difíciles que, si bien pueden llevar a la obtención de soluciones no óptimas, pueden resolver estos problemas en términos de satisfacción para el usuario y con un gasto aceptable de recursos

computacionales. Estas técnicas se conocen como heurísticas. Siempre existe la posibilidad de que alguien pruebe que "P = NP", pero la posibilidad parece ser demasiado pequeña. Hasta que alguien encuentre esta prueba, el uso de heurísticas tiene una justificación considerable.

2.2. Heurísticas y Metaheurísticas

2.2.1. Definiendo Heurística

George Polya define *heurística* como "el estudio de los métodos y reglas de descubrimiento e invención" [218]. Este significado puede ser rastreado hasta su raíz, el verbo griego *eurisco*, que significa, "yo descubro". Cuando Arquímedes salió de su famoso baño sujetando la corona de oro, gritó "¡Eureka!" que significa "¡Lo encontré!". Difícilmente se podría considerar a Arquímedes como el padre de las heurísticas, sin embargo, y de hecho, la perspectiva etimológica no ayuda demasiado, como White ha apuntado [258]. La palabra ha sido utilizada en los círculos de Inteligencia Artificial con una connotación totalmente distinta. Pearl [212], por ejemplo, utiliza el término para incluir métodos tales como ramificación y poda que encuentran soluciones óptimas globales, y este uso probablemente es previo al ahora más común. De hecho, este uso es perfectamente razonable; si una heurística es un método de encontrar, estamos autorizados a preguntar ¿que está encontrando si no es un óptimo global?. Lo que en investigación operativa es casi universalmente llamado heurística sería descrito mejor como un método de búsqueda, puesto que no puede garantizar que encontrará algo.

Sin embargo, en el uso que se ha hecho más común el término *heurística* se usa como un contraste con métodos que garantizan encontrar un óptimo global. El primer uso documentado del término en el *International Abstracts of OR* parece datar del 1960 (en un artículo escrito en francés), pero a mediados de los 1960s, la connotación actual parece haberse establecido.

Existen gran cantidad de definiciones de heurística, como la anteriormente mencionada de Polya. Una de las más extendidas es la propuesta en [230]: "Una heurística es una técnica que busca soluciones buenas (es decir cercanas al óptimo) con un coste computacional razonable sin ser capaz de garantizar su factibilidad o optimalidad, o incluso en muchos casos de determinar cómo de cerca a la optimalidad se encuentra una solución factible".

Desde su aparición se han propuesto gran cantidad de métodos heurísticos para resolver problemas de optimización. Las causas de esta explosión de interés parecen tener su causa en el desarrollo del concepto de complejidad computacional, que ha proporcionado una base racional para explorar técnicas heurísticas en lugar de intentar encontrar la optimalidad.

Muchas heurísticas son especificas del problema que se desea resolver, de tal manera que

el método que funciona para un problema no puede ser usado para resolver otro distinto. Sin embargo resulta más interesante la búsqueda de métodos aplicables en un modo más general. De esta manera surgen las metaheurísticas, que intentan extraer lo mejor de los métodos heurísticos y emplearlo en contextos más extensos.

2.2.2. Metaheurísticas

El término metaheurística fue acuñado por F. Glover, en el año 1986 [126]. Etimológicamente, deriva de dos palabras griegas, que son meta y heurística. El segundo término, ya explicado anteriormente, y el prefijo meta se podría traducir como "más allá de" o "en un nivel superior". Con estos términos, Glover [127] pretendía definir, métodos de resolución que orquestan una interacción entre procedimientos de mejora local y estrategias de alto nivel, para crear un proceso capaz de escapar de los óptimos locales y realizar una búsqueda robusta del espacio de soluciones. A lo largo del tiempo, la definición se ha ampliado para incluir cualquier procedimiento que emplea estrategias para escapar de la optimalidad local en espacios de soluciones complejos.

En la literatura se pueden encontrar distintas definiciones del término metaheurística [188, 208, 253]. Tomando la definición de [188],

> *Las metaheurísticas son estrategias inteligentes para diseñar o mejorar procedimientos heurísticos muy generales con un alto rendimiento,*

podemos apreciar que el concepto de metaheurística se basa en las diferentes interpretaciones de como resolver un problema de manera inteligente.

La evolución de las metaheurísticas durante los últimos años ha tenido un crecimiento prácticamente exponencial. Esta relevancia se refleja en la publicación de volúmenes especiales sobre metaheurísticas que aparecen en diversas colecciones editoriales o revistas periódicas de los campos de Investigación Operativa, Inteligencia Artificial, Ingeniería y Ciencias de la Computación (*Journal of Heuristics* [173], *INFORMS Journal of Computing* [71], los libros [78, 127, 231], los congresos temáticos MIC, MAEB). Se destacan las publicaciones [32, 188] en las que se exponen las principales propiedades y las propiedades deseables de una metaheurística.

Principales propiedades de una metaheurística, [32]:

- Son estrategias que guían el proceso de búsqueda.
- Su objetivo es explorar eficientemente el espacio de búsqueda para encontrar las soluciones sub-óptimas.

- Las técnicas que la forman van desde algoritmos de búsqueda simple a complejos procesos de aprendizaje.
- Son algoritmos aproximados y generalmente no deterministas.
- Incorporan mecanismos para escapar cuando están confinados en áreas del espacio de búsqueda.
- Sus conceptos básicos permiten una descripción de nivel abstracto.
- No son específicas del problema.
- Pueden hacer uso del conocimiento específico del dominio y/o de la experiencia de la búsqueda para influenciar la búsqueda.
- Generalmente se basan en la experiencia y en conocimientos empíricos.

Propiedades deseables en una metaheurística, [188]:

- Simple. La metaheurística debe estar basada en un principio sencillo y claro; fácil de comprender.
- Precisa. Los pasos y fases de la metaheurística deben estar formulados en términos concretos.
- Coherente. Los elementos de la metaheurística deben deducirse naturalmente de sus principios.
- Efectiva. Los algoritmos derivados de la metaheurística deben proporcionar soluciones de muy alta calidad; óptimas o muy cercanas a las óptimas.
- Eficaz. La probabilidad de alcanzar soluciones óptimas de casos realistas con la metaheurística debe ser alta.
- Eficiente. La metaheurística debe realizar un buen aprovechamiento de recursos computacionales; tiempo de ejecución y espacio de memoria.
- General. La metaheurística debe ser utilizable con buen rendimiento en una amplia variedad de problemas.
- Adaptable. La metaheurística debe ser capaz de adaptarse a diferentes contextos de aplicación y modificaciones importantes del modelo.
- Robusta. El comportamiento de la metaheurística debe ser poco sensible a pequeñas alteraciones del modelo o contexto de aplicación.
- Interactiva. La metaheurística debe permitir que el usuario pueda aplicar sus conocimientos para mejorar el rendimiento del procedimiento.
- Múltiple. La metaheurística debe suministrar diferentes soluciones alternativas de alta calidad entre las que el usuario pueda elegir.

- Autónoma. La metaheurística debe permitir un funcionamiento autónomo, libre de parámetros que se puedan establecer automáticamente.

Es importante tener en cuenta que estas estrategias deben ser escogidas de manera tal que haya un balance dinámico entre la explotación de la experiencia acumulada en la búsqueda de las soluciones (*intensificación*) y la exploración del espacio de búsqueda (*diversificación*). Esto es necesario, por un lado, para identificar rápidamente las regiones en el espacio de búsqueda que tienen soluciones de alta calidad; y por otro, no derrochar demasiado tiempo en regiones del espacio de búsqueda que ya hayan sido exploradas o no brinden soluciones de alta calidad.

2.3. Clasificación de las Metaheurísticas

En [188] aparece una interesante clasificación de las metaheurísticas. Las metaheurísticas son estrategias para diseñar procedimientos heurísticos, por tanto, los tipos de metaheurísticas se establecen, en primer lugar, en función del tipo de procedimientos a los que se refiere. Algunos de los tipos fundamentales son las metaheurísticas para los métodos de relajación, las metaheurísticas para los procesos constructivos, las metaheurísticas para las búsquedas por entornos y las metaheurísticas para los procedimientos evolutivos.

1. Metaheurísticas de relajación

   Las metaheurísticas de relajación se refieren a procedimientos de resolución de problemas que utilizan relajaciones del modelo original (es decir, modificaciones del modelo que hacen al problema más fácil de resolver), cuya solución facilita la solución del problema original. Una relajación de un problema es un modelo simplificado obtenido al eliminar, debilitar o modificar restricciones (u objetivos) del problema real.

2. Metaheurísticas constructivas

   Las metaheurísticas constructivas se orientan a los procedimientos que tratan de la obtención de una solución a partir del análisis y selección paulatina de las componentes que la forman. Las metaheurísticas constructivas aportan soluciones del problema por medio de un procedimiento que incorpora iterativamente elementos a una estructura, inicialmente vacía, que representa a la solución, y además, establecen estrategias para seleccionar las componentes con las que se construye una buena solución del problema.

3. Metaheurísticas de búsqueda

   Las metaheurísticas de búsqueda guían los procedimientos que usan transformaciones o movimientos para recorrer el espacio de soluciones alternativas y explotar las estructuras de entornos asociadas.

Unas de las más importante es son las que establecen estrategias para recorrer el espacio de soluciones del problema transformando de forma iterativa soluciones de partida.

4. Metaheurísticas evolutivas

   Las metaheurísticas evolutivas están enfocadas a los procedimientos basados en conjuntos de soluciones que evolucionan sobre el espacio de soluciones.

   Las metaheurísticas evolutivas establecen estrategias para conducir la evolución en el espacio de búsqueda de conjuntos de soluciones (usualmente llamados poblaciones) con la intención de acercarse a la solución óptima con sus elementos. El aspecto fundamental de estas metaheurísticas evolutivas consiste en la interacción entre los miembros de la población frente a las búsqueda que se guían por la información de soluciones individuales.

   Las diferentes metaheurísticas evolutivas se distinguen por la forma en que combinan la información proporcionada por los elementos de la población para hacerla evolucionar mediante la obtención de nuevas soluciones. Los algoritmos genéticos y meméticos, y los de estimación de distribuciones emplean fundamentalmente procedimientos aleatorios, mientras que las metaheurísticas de búsqueda dispersa o de re-encadenamiento de caminos emplean procedimientos sistemáticos.

Si se observan a las metaheurísticas desde un nivel abstracto y se toman como "puras" quizás se puedan establecer algunas clasificaciones teniendo en cuenta sus mecanismos, sus principios, etc. Pero en la práctica es diferente, ya que al ser éstas desmontables de manera relativamente fácil en sus componentes y además brindar varias vías para reensamblarlas, propicia la hibridación entre ellas. Por ello, se hace difícil una clasificación en la que claramente se puedan insertar a todas.

Hay varias maneras de clasificar a las metaheurísticas, entre las más conocidas están las siguientes: en función del origen del método, se habla de algoritmos "bioinspirados" (algoritmos genéticos, de colonia de hormigas, etc) vs. "no bioinspirados". Otra posibilidad es categorizarlos en función de la utilización o no de memoria, o en función del uso de una función objetivo estática o dinámica. Una clasificación interesante es la que surge al diferenciar los métodos que mantienen una única solución, frente a los que mantienen un conjunto o población de soluciones.

Aquí se ha tomado la clasificación [78] basada en métodos de trayectoria y métodos basados en poblaciones ya que da una descripción más clara de cada tipo de metaheurística:

1. Metaheurísticas basadas en trayectorias

   Son aquellas metaheurísticas que establecen estrategias para recorrer el espacio de soluciones del problema transformando de forma iterativa soluciones de partida.

   Una búsqueda sobre un espacio, consiste en generar una sucesión de puntos del espacio pasando de uno a otro por medio de una serie de transformaciones o movimientos, como si siguiera una trayectoria. Un procedimiento de búsqueda para resolver un problema de optimización realiza recorridos sobre el entorno o espacio de las soluciones alternativas y selecciona la mejor solución encontrada en el recorrido.

   El esquema general de un procedimiento de búsqueda basado en trayectorias o por entornos consiste en generar una solución inicial y seleccionar iterativamente un movimiento para modificar la solución hasta que se cumpla el criterio de parada. Las soluciones son evaluadas mientras se recorren y se propone la mejor solución del problema encontrada.

   Algunas de las metaheurísticas de esta clasificación son Temple Simulado [97], Búsqueda Tabú [129], Búsqueda por Entornos Variables [141], Búsqueda Local Iterativa [185] y Búsqueda Local Guiada [247].

2. Metaheurísticas basadas en poblaciones

   Las metaheurísticas basadas en poblaciones son las que conducen la evolución en el espacio de búsqueda de conjuntos de soluciones, usualmente llamados poblaciones, con la intención de acercarse a la solución óptima con sus elementos. El aspecto fundamental de estas heurísticas consiste en la interacción entre los miembros de la población frente a las búsquedas que se guían por la información de soluciones individuales.

   Se distinguen por la forma en que combinan la información proporcionada por los elementos de la población para hacerla evolucionar mediante la obtención de nuevas soluciones. En una búsqueda en grupo o basada en poblaciones se sustituye la solución actual que recorre el espacio de soluciones, por un conjunto de soluciones que lo recorren conjuntamente interactuando entre ellas. Además de los movimientos aplicables a las soluciones que forman parte de este conjunto, denominado grupo o población de búsqueda, se contemplan otros operadores para generar nuevas soluciones a partir de las ya existentes.

   Algunas de las metaheurísticas de esta clasificación son Algoritmos Genéticos [16], Algoritmos Meméticos [82] y Colonia de Hormigas [100].

2.4. Algoritmos metaheurísticos

2.4.1. Introducción

Como hemos definido anteriormente, denotamos $x \in I_P$ como una instancia de un problema de optimización P. Se define la estructura de vecindario $\mathcal{N}$ como una aplicación $\mathcal{N} : SOL_P(x) \rightarrow 2^{SOL_P(x)}$, la cual determina para cada solución $y \in SOL_P(x)$ el conjunto $\mathcal{N}(y) \subseteq SOL_P(x)$ de soluciones "cercanas" (en algún sentido) a y.

El conjunto $\mathcal{N}(y)$ se denomina el vecindario de la solución y. Ejemplos de vecindarios surgen naturalmente, por ejemplo, si se utiliza una función de distancia $dist : SOL_P(x) \times SOL_P(x) \rightarrow \mathbb{R}$. Se puede definir el vecindario de una solución y como, $\mathcal{N}(y) = \{z \in SOL_P(x) \ / \ dist(y, z) \leq \epsilon\}$.

En general, las soluciones z se obtienen a partir de la aplicación de un operador $\mathcal{O}$, al que se suele denominar "Movimiento", que modifica en algún sentido la solución actual y para obtener nuevas soluciones. Este operador suele incluir algún procedimiento de aleatorización, con lo cual sucesivas aplicaciones de $\mathcal{O}(y)$ permiten obtener soluciones z diferentes.

El concepto de vecindad está presente en todos los métodos heurísticos a la hora de resolver un problema de optimización. A continuación se presentan algunos de los algoritmos metaheurísticos más utilizados en la literatura. Sin embargo, esta revisión no pretende ser exhaustiva, puesto que queda al margen del enfoque de este trabajo.

2.4.2. Metaheurísticas basadas en trayectorias

Una metaheurística basada en trayectorias sobre un espacio de búsqueda consiste en la generación de una sucesión de puntos del espacio pasando de uno a otro por medio de una serie de movimientos, como si siguiera una trayectoria. El esquema general de una metaheurística basada en trayectorias consiste en la generación de una solución inicial y selecciona iterativamente un movimiento para modificar la solución hasta que se cumpla el criterio de parada. Las soluciones son evaluadas mientras se recorren y se propone la mejor solución del problema encontrada. Destacamos las siguientes: Temple Simulado, Búsqueda Tabú y Búsqueda por Entornos Variables.

2.4.2.1. Temple Simulado

El Temple o Enfriamiento Simulado (*Simulated Annealing, SA*), [97, 145] es conocida como la primera de las metaheurísticas y uno de los primeros algoritmos con una estrategia explícita para escapar de óptimos locales. Su nombre hace referencia al proceso físico en el

que está basado. Su implementación es la mezcla lógica de dos conceptos: Hill Climbing y aleatoriedad. Los algoritmos de Hill Climbing son muy propensos a estancarse en óptimos locales. Por otro lado, un algoritmo puramente aleatorio asegura que todas las soluciones pueden ser tomadas, pero es gravemente ineficaz. SA combina estos dos conceptos obteniendo un algoritmo de Hill Climbing estocástico.

El SA es el exponente más importante del tipo de metaheurísticas donde la probabilidad de aceptación es una función del empeoramiento producido. Para evitar los óptimos locales permite algunos movimientos hacia soluciones peores controlando su frecuencia mediante una función que disminuye conforme avanza la búsqueda.

El esquema básico, supuesto un problema de maximización, se presenta en Algoritmo 1.

Algoritmo 1: Temple simulado

```
begin
    k=0;
    t_0=T;
    s_a = solucion-inicial();
    while no-finalizacion do
        s_v = GenerarVecino(s_a);
        if (f(s_v)  f(s_a)) > 0 then
            s_a = s_v;
        else
            ρ = rand(0,1);
            if ρ ≤ e^((f(s_v)  f(s_a))/t_k) then
                s_a = s_v;
            end
        end
        k=k+1;
        t_k=reducirTemperatura(t_k  1);
    end
    devolver s_a
end
```

Donde T representa la temperatura inicial, t_k la temperatura en el momento k y $f(s)$ la función objetivo que mide el valor de bondad de la solución s. La función "generarVecino(s)" devuelve un vecino $s_v \in \mathcal{N}(s)$ generado aleatoriamente, la función "rand(0,1)" devuelve un número aleatorio entre 0 y 1, y "reducirTemperatura" es una función decreciente del valor de temperatura.

La estrategia del SA comienza con una temperatura inicial "alta", lo cual proporciona una probabilidad también alta de aceptar movimientos de empeoramiento. En cada iteración se

reduce la temperatura y por lo tanto las probabilidades de aceptar soluciones peores también disminuyen. En cada iteración se genera un número concreto de vecinos, $L(t) \subset \mathcal{N}(t)$, que puede ser fijo para toda la ejecución o depender de la iteración concreta. Cada vez que se genera un vecino, se aplica el criterio de aceptación para ver si sustituye a la solución actual.

Díaz et al. [97] plantean que su principal desventaja está en que puede necesitar mucho tiempo de cálculo cuando los problemas son grandes y complejos. Como ello está relacionado directamente con el tamaño del espacio de soluciones, se trata de reducir éste, que podría ser a través de un preprocesamiento mediante un conjunto de reglas de reducción, o reduciendo directamente la proporción del entorno a ser explorado en cada iteración.

Se ha aplicado a varios problemas de optimización en los que destacan los problemas de localización, empaquetado, planificación, asignación, y localización entre otros. Una recopilación sobre esto se puede ver en [97, 174, 265].

2.4.2.2. Búsqueda Tabú

La Búsqueda Tabú (*Tabu Search, TS*) [128, 129], es una de las estrategias que tratan de utilizar la memoria del proceso de búsqueda para mejorar su rendimiento evitando que la búsqueda se concentre en una misma zona del espacio. El marco de memoria de TS explota la historia del proceso de resolución del problema haciendo referencia a cuatro dimensiones principales consistentes en la propiedad de ser reciente, en frecuencia, en calidad, y en influencia. Además crea ciertas estructuras para hacer posible tal explotación.

TS está fundamentada en las ideas expuestas por F. Glover en prohibir temporalmente soluciones muy parecidas a las últimas soluciones del recorrido. En el origen del método el propósito era sólo evitar la reiteración en una misma zona de búsqueda recordando las últimas soluciones recorridas. Sin embargo, posteriormente se han realizado diversas propuestas para rentabilizar la memoria a medio o largo plazo.

La TS se caracteriza por dos aspectos principales:

- Presenta un mecanismo de generación de vecinos modificado que evita la exploración de zonas del espacio de búsqueda que ya han sido visitadas: generación de entornos tabú restringidos.
- Emplea mecanismos para mejorar la capacidad del algoritmo para la exploración-explotación del espacio de búsqueda.

Para realizar las dos tareas anteriores, hace uso de unas estructuras de memoria adaptables:

- Memoria de corto plazo o Lista tabú, que permite guiar la búsqueda de forma inmediata,

desde el comienzo del procedimiento (generación de entornos tabú restringidos).

- Memoria de largo plazo, que guarda información que permite guiar la búsqueda a posteriori, después de una primera etapa en la que se ha realizado una o varias ejecuciones del algoritmo en las que se ha aplicado la memoria a corto plazo.

La información guardada en esta memoria se usa para dos tareas distintas:

- Intensificar la búsqueda, consiste en regresar a buenas regiones ya exploradas para estudiarlas más a fondo. Para ello se favorece la aparición de aquellos atributos asociados a buenas soluciones encontradas.
- Diversificar la búsqueda, consiste en visitar nuevas áreas no exploradas del espacio de soluciones. Para ello se modifican las reglas de elección para incorporar a las soluciones atributos que no han sido usados frecuentemente.

El esquema básico de una TS se muestra en el Algoritmo 2.

Algoritmo 2: Búsqueda Tabú

```
begin
    s_a = solucion-inicial();
    ListaTabu = { };
    while no-finalizacion do
        s_v = Mejorar(s_a,N(s_a),ListaTabu);
        s_a = s_v;
        Actualizar(ListaTabu);
    end
    devolver s_a
end
```

La estrategia comienza desde una solución inicial s_a, construye un entorno $\mathcal{N}(s_a)$ y selecciona la mejor solución s_v que pertenece a $\mathcal{N}(s_a)$. El procedimiento continua hasta que cierta condición de parada se verifique. En ciertos momentos del proceso de búsqueda algunos de los movimientos se introducen en la lista tabú, permaneciendo sólo durante un cierto número de iteraciones, aunque hay ciertas excepciones. Cuando un movimiento tabú proporciona una solución mejor que cualquier otra previamente encontrada, su clasificación tabú puede eliminarse. La condición que permite dicha eliminación se denomina *criterio de aspiración*.

Las restricciones tabú y el criterio de aspiración juegan un papel dual en la restricción y guía del proceso de búsqueda. Las restricciones tabú, permiten que un movimiento sea admisible si no está clasificado como tabú, mientras que si el criterio de aspiración se satisface, permite que un movimiento sea admisible aunque esté clasificado como tabú.

La longitud de la lista tabú es un parámetro. Su definición no es trivial ya que si es demasiado pequeño pueden ocurrir ciclos, mientras que si es demasiado grande, puede restringir en exceso la búsqueda. Esta lista recibe también el nombre de memoria de corto plazo. En ocasiones, y como base para las estrategias de exploración/explotación, también se utilizan memorias de término intermedio y largo plazo. Por ejemplo, la memoria basada en frecuencia se usa como parte de una estrategia de largo plazo.

La memoria basada en frecuencia proporciona un tipo de información que complementa la proporcionada por la memoria basada en lo reciente, ampliando la base para seleccionar movimientos preferidos. La memoria basada en frecuencia típicamente encuentra su uso más productivo como parte de una estrategia de periodo más largo.

De sus áreas de aplicación tradicionales como: problemas de teoría de grafos, localización y asignación, planificación y enrutamiento se ha movido a otras nuevas: optimización continua, optimización multi-criterio, programación estocástica, programación entera mixta, problemas de decisión en tiempo real, etc, [44, 128].

2.4.2.3. Búsqueda por Entornos Variables

Las metaheurísticas de entorno variable modifican de forma sistemática el tipo de movimiento con el objeto de evitar que la búsqueda se quede atrapada en óptimos locales debido a una estructura de entornos rígida.

La Búsqueda por Entornos Variables, *(Variable Neighbourhood Variable, VNS)* [139, 141, 142], es una metaheurística que está basada en un principio simple: cambiar sistemáticamente de estructura de entornos dentro de la búsqueda para escapar de los mínimos locales.

La VNS básico obtiene una solución del entorno de la solución actual, ejecuta una búsqueda monótona local desde ella hasta alcanzar un óptimo local, que reemplaza a la solución actual si ha ocurrido una mejora y modifica la estructura de entorno en caso contrario. Un pseudocódigo de este esquema se encuentra en el Algoritmo 3.

La VNS está basada en tres hechos simples:

- Un mínimo local con una estructura de entornos no lo es necesariamente con otra.
- Un mínimo global es mínimo local con todas las posibles estructuras de entornos.
- Para muchos problemas, los mínimos locales con la misma o distinta estructura de entornos están relativamente cerca.

Esta última observación, que es empírica, implica que los óptimos locales proporcionan información acerca del óptimo global. Por ejemplo, puede ser que ambas soluciones tengan ca-

Algoritmo 3: Búsqueda por Entornos Variables (VNS)

```
begin
    repeat
        k=1;
        repeat
            s_v = ObtenerVecino(N_k(s_a));
            sol = BusquedaLocal(s_v);
            if sol es mejor que s_a then
                s_a = sol
            else
                k=k+1
            end
        until k = k_max;
    until finalizacion;
    devolver s_a
end
```

racterísticas comunes. Sin embargo, generalmente no se conoce cuáles son esas características. Es procedente, por tanto, realizar un estudio organizado en las proximidades de este óptimo local, hasta que se encuentre uno mejor.

Estos hechos sugieren el empleo de varias estructuras de entornos en las búsquedas locales para abordar un problema de optimización. El cambio de estructura de entornos se puede realizar de forma determinística, estocástica, o determinística y estocástica a la vez.

Se le han hecho extensiones que constituyen mejoras prácticas de la VNS que han permitido resolver con éxito problemas muy grandes [141].

La VNS se ha aplicado a diversos problemas como los problemas de satisfacción, el aprendizaje en redes bayesianas, y los de clasificación y planificación, problemas de empaquetado, localización y de rutas, [140, 141, 195].

2.4.3. Metaheurísticas basadas en poblaciones

Las metaheurísticas basadas en poblaciones son las que conducen la evolución en el espacio de búsqueda de conjuntos de soluciones, usualmente llamados poblaciones, con la intención de acercarse a la solución óptima. El aspecto fundamental de estas metaheurísticas consiste en la interacción entre los miembros de la población frente a las búsquedas que se guían por la información de soluciones individuales.

Se distinguen por la forma en que combinan la información proporcionada por los elementos de la población para hacerla evolucionar mediante la obtención de nuevas soluciones.

En una búsqueda en grupo o basada en poblaciones se sustituye la solución actual que recorre el espacio de soluciones, por un conjunto de soluciones que lo recorren conjuntamente interactuando entre ellas. Además de los movimientos aplicables a las soluciones que forman parte de este conjunto, denominado grupo o población de búsqueda, se contemplan otros operadores para generar nuevas soluciones a partir de las ya existentes. Destacamos las siguientes: Algoritmos Genéticos, Enjambres de partículas y Colonias de Hormigas.

2.4.3.1. Algoritmos Genéticos

Los Algoritmos Genéticos *(Genetic Algorithms, GA)* son un campo de investigación interdisciplinario con relaciones con biología, inteligencia artificial y optimización numérica, y han sido utilizados en varias disciplinas científicas e ingenieriles. Los Algoritmos Genéticos operan sobre una población de soluciones potenciales, llamadas cromosomas o individuos, aplicando el principio de supervivencia del más adaptado para producir aproximaciones cada vez mejores a una solución sub-óptima. En cada generación, se crea un conjunto nuevo de soluciones usando un proceso de selección de individuos de acuerdo a su nivel de adecuación al dominio del problema y un conjunto de operadores genéticos (cruce y mutación). Este proceso lleva a la evolución de las poblaciones de individuos que están cada vez más adaptadas a su entorno.

Mediante una imitación del mecanismo genético de los organismos biológicos, los GAs exploran el espacio de soluciones asociado a un determinado problema. Se establece una codificación apropiada de las soluciones del espacio de búsqueda y una forma de evaluar la función objetivo para cada una de estas codificaciones. Las soluciones se identifican con individuos que pueden formar parte de la población de búsqueda. La codificación de una solución se interpreta como el *cromosoma* del individuo compuesto de un cierto número de *genes* a los que les corresponden ciertos *alelos*. El gen es la característica del problema; alelo, el valor de la característica; *genotipo*, la estructura y *fenotipo*, la estructura sometida al problema. Cada cromosoma es una estructura de datos que representa una de las posibles soluciones del espacio de búsqueda del problema. Los cromosomas son sometidos a un proceso de evolución que envuelve evaluación, selección, recombinación sexual o cruce y mutación. Después de varios ciclos de evolución la población deberá contener individuos más aptos. En el Algoritmo 4 se presenta el esquema básico del GA.

En este algoritmo 4 se consideran dos operaciones básicas: la mutación y el cruce. La mutación de un individuo consiste en modificar uno o varios genes cambiando al azar el alelo correspondiente. El cruce de dos individuos, llamados padres, produce un individuo hijo tomando un numero k (elegido al azar) de genes de uno de los padres y los $t \quad k$ del otro.

Algoritmo 4: Algoritmo Genético

```
begin
    t = 0;
    inicializar(P(t));
    evaluar(P(t));
    while no-finalizacion do
        t = t + 1;
        Seleccionar P(t) desde P(t-1);
        P(t)=Cruzar(P(t));
        P(t)=Mutar(P(t));
        evaluar(P(t));
    end
    devolver mejor solucion;
end
```

La población evoluciona de acuerdo a las estrategias de selección de individuos, tanto para las operaciones como para la supervivencia. La selección se puede hacer simulando una lucha entre los individuos de la población con un procedimiento que, dados dos individuos selecciona uno de ellos teniendo en cuenta su valoración (la función objetivo) y la adaptación al ambiente y a la población (criterios de diversidad, representatividad).

Para construir un GA se necesita:

- Una representación de las soluciones del problema.
- Una forma de crear una población inicial de soluciones.
- Una función de evaluación en términos de adaptación de la solución evaluada.
- Operadores genéticos.
- Una forma para seleccionar los individuos para ser padres.
- Una forma de cómo reemplazar a los individuos.
- Una condición de parada.

Profundizamos en algunas de las operaciones necesarias en la realización de un GA.

- *Evaluación:* Es dar a cada configuración ("individuo") un peso de importancia con respecto a los demás. Fundamentalmente se tiene en cuenta el valor de la configuración con respecto a la función objetivo.
- *Selección:* La selección significa elegir de la población las configuraciones que servirán de base para generar la población siguiente. En la estrategia de selección se debe garantizar que los mejores individuos tengan una mayor posibilidad de ser padres

(reproducirse) frente a los individuos menos buenos, pero también ser cuidadosos para dar una posibilidad de reproducirse a los individuos menos buenos. Estos pueden incluir material genético útil en el proceso de reproducción. Esta idea define la presión selectiva que conducirá la reproducción como la selección fuerte de los mejores.

- *Cruce:* En el *cruce* o *crossover* nuevos individuos son generados a partir de la combinación de dos o más individuos. La idea intuitiva del operador de cruce es permitir el intercambio de información entre individuos de la población. Evidentemente, es la existencia del cruce lo que gobierna la eficiencia de los GA y los destaca nítidamente de otro tipo de estrategias. En [172] se describen diversos operadores genéticos de cruce.
- *Mutación:* En la *mutación* nuevos individuos son generados a partir de pequeños cambios en un individuo. La mutación introduce una variación aleatoria sobre subconjuntos seleccionados de la población, generalmente de un elemento. La mutación permite incorporar nueva información que no se puede generar a partir de la clausura transitiva del cruce sobre la población. Para representar la mutación pueden tenerse uno o más operadores. Hay que tener en cuenta que debe permitir alcanzar cualquier parte del espacio de búsqueda, su tamaño debe ser controlado y debe producir cromosomas válidos, o sea, acordes al problema.
- *Reemplazo:* En la estrategia de reemplazamiento la presión selectiva se ve también afectada por la forma en que los cromosomas de la población son reemplazados por los nuevos descendientes. Se usan métodos de reemplazamiento aleatorios, o determinísticos. También se puede decidir no reemplazar al mejor(es) cromosoma(s) de la población (Elitismo).

Los GAs han sido aplicados en diversos problemas de optimización, tales como: optimización de funciones matemáticas, optimización combinatoria, optimización de planeamiento, problema del viajante, problema de optimización de rutas, optimización de diseño de circuitos, optimización de distribución, optimización en negocios, etc. En [132, 174, 229] aparecen muchos ejemplos de sus aplicaciones.

2.4.3.2. Enjambre de Partículas

Optimización con enjambre de partículas (*Particle Swarm Optimization, PSO*) es una metaheurística evolutiva inspirada en el comportamiento social de las bandadas de pájaros o bancos de peces desarrollada por Eberhart y Kennedy en 1995 [109]. Comparte similaridades con técnicas de la computación evolutiva, como los GAs.

En PSO las soluciones, llamadas partículas, se mueven en el espacio de búsqueda guiadas

por la partícula que mejor solución ha encontrado hasta el momento y que hace de líder de la bandada. Cada partícula evoluciona teniendo en cuenta la mejor solución encontrada en su recorrido y al líder (a diferencia de los GAs no se utilizan operadores de cruce y mutación en este proceso de evolución). En cada iteración, las partículas modifican su velocidad hacia la mejor solución de su entorno teniendo en cuenta la información del líder. En el Algoritmo 5 se muestra el algoritmo PSO.

Algoritmo 5: Enjambre de partículas

```
begin
    foreach particula p do
        inicPartic(p);
    end
    while terminacion do
        foreach particula p do
            fitn=calcFitness(p);
            if fitn es mejor que la mejor partícula pBest then
                pBest = p;
            end
        end
        gBest = particulaMejor(pBest);
        foreach particula p do
            calcVeloc(p);
            actualizaPosit(p);
        end
    end
    devolver mejor solución
end
```

En cada iteración, cada partícula se actualiza por dos valores. El primero es la mejor solución que esta partícula ha logrado, este valor se llama *pbest*; el segundo es el mejor obtenido por cualquier partícula en toda la población, llamado *gbest*. Cuando una partícula toma parte de la población de un vecindario topológico, el mejor valor en este vecindario es llamado *lbest*, es decir, se crea un vecindario para cada partícula conteniendo los k vecinos más cercanos en la población.

El concepto de PSO consiste en ir cambiando la velocidad (aceleración) de cada partícula hacia su *pbest* y *lbest*. Después de encontrar los dos mejores valores, la partícula actualiza su velocidad y posición con las siguientes ecuaciones:

$$v[\cdot] = w \cdot v[\cdot] + c_1 \times rand(\cdot) \times (pbest[\cdot] \quad present[\cdot]) + c_2 \times rand(\cdot) \times (gbest[\cdot] \quad present[\cdot])$$

$$present[\cdot] = present[\cdot] + v[\cdot]$$

donde,

- $v[\cdot]$ es la velocidad de la partícula,
- $present[\cdot]$ es la partícula actual,
- $pbest[\cdot]$ y $gbest[\cdot]$ fueron definidos arriba,
- $rand(\cdot)$ es un número aleatorio entre (0,1),
- c_1 y c_2 son factores de aprendizaje.

La suma de aceleraciones podría causar que la velocidad exceda V_{max}, el cual es un parámetro especificado por el usuario, por ello la velocidad se limita a V_{max}.

PSO obtiene muy rápido sus resultados y además tiene pocos parámetros para ajustar por lo que resulta atractivo para muchas áreas de aplicación e investigación [109,189] tales como: optimización, entrenamiento de redes neuronales, sistemas de control difusos, etc.

2.4.3.3. Colonia de Hormigas

La metaheurística de Sistema o Colonia de Hormigas para Optimización *(Ants Colony Optimization, ACO)* [96, 100] emplea estrategias inspiradas en el comportamiento de las colonias de hormigas para descubrir fuentes de alimentación, al establecer el camino más corto entre éstas y el hormiguero y transmitir esta información al resto de sus compañeras.

Un aspecto interesante del comportamiento de muchas especies de hormigas es su habilidad para encontrar los caminos más cortos entre su hormiguero y las fuentes de alimento. Este hecho es especialmente interesante si se tiene en cuenta que muchas de las especies de hormigas son casi ciegas, lo que evita el uso de pistas visuales. Mientras que se mueven entre el hormiguero y la fuente de alimento, algunas especies de hormigas depositan una sustancia química denominada feromona (una sustancia que puede "olerse"). Si no se encuentra ningún rastro de feromona, las hormigas se mueven de manera básicamente aleatoria, pero cuando existe feromona depositada, tienen mayor tendencia a seguir el rastro. De hecho, los experimentos realizados por biólogos han demostrado que las hormigas prefieren de manera probabilística los caminos marcados con una concentración superior de feromona. En la práctica, la elección entre distintos caminos toma lugar cuando varios caminos se cruzan. Entonces, las hormigas eligen el camino a seguir con una decisión probabilística sesgada por la cantidad de feromona: cuanto más fuerte es el rastro de feromona, mayor es la probabilidad de elegirlo. Puesto que las hormigas depositan feromona en el camino que siguen, este comportamiento lleva a un proceso de autorefuerzo que concluye con la formación de rastros señalados por una concentración de feromona elevada. Este comportamiento permite además a las hormigas encontrar los caminos más cortos entre su hormiguero y la fuente del alimento.

Esta convergencia se complementa con la acción del entorno natural que provoca que la feromona se evapore transcurrido un cierto tiempo. Así, los caminos menos prometedores pierden progresivamente feromona porque son visitados cada vez por menos hormigas.

Cada hormiga construye una solución, o un componente de ésta, comenzando en un estado inicial seleccionado de acuerdo a criterios dependientes del problema. Mientras construye su propia solución colecciona información sobre las características del problema y sobre su actuación y usa esta información para modificar la representación del problema. Las hormigas pueden actuar de forma concurrente e independiente mostrando un comportamiento cooperativo, pero sin una comunicación directa. Una simple hormiga debe ser capaz por sí sola de encontrar una solución (probablemente de baja calidad). Las soluciones de alta calidad son sólo encontradas como resultado de la cooperación global entre todos los agentes de la colonia trabajando concurrentemente diferentes soluciones.

Los algoritmos de ACO son esencialmente algoritmos constructivos: en cada iteración del algoritmo, cada hormiga construye una solución al problema recorriendo un grafo de construcción. Cada arista del grafo, que representa los posibles pasos que la hormiga puede dar, tiene asociada dos tipos de información que guían el movimiento de la hormiga:

- Información heurística, que mide la preferencia heurística de moverse desde el nodo r hasta el nodo s, o sea, de recorrer la arista a_{rs}. Las hormigas no modifican esta información durante la ejecución del algoritmo.
- Información de los rastros de feromona artificiales, que mide la "deseabilidad aprendida" del movimiento de r a s. Imita a la feromona real que depositan las hormigas naturales. Esta información se modifica durante la ejecución del algoritmo dependiendo de las soluciones encontradas por las hormigas.

Las decisiones acerca de cuándo las hormigas deberían liberar feromona sobre el ambiente y cuánta feromona debería depositarse dependen de las características del problema y del diseño de la implementación. Las hormigas pueden liberar feromona mientras construyen la solución, o después de que la solución ha sido construida o ambos.

El algoritmo básico de ACO se muestra en 6.

El procedimiento principal de la metaheurística ACO controla, mediante **PlanifActiv**, la planificación de las tres componentes siguientes:

1. gxActivHormigas: La generación y puesta en funcionamiento de las hormigas artificiales.
2. evapFeromonas: La evaporación de feromona. La evaporación de feromona la lleva a cabo el entorno y se usa como un mecanismo que evita el estancamiento en la búsqueda y permite que la hormigas busquen y exploren nuevas regiones del espacio.

Algoritmo 6: Colonia de Hormigas

```
begin
    while terminacion do
        PlanifActiv
            gxActivHormigas();
            evapFeromonas();
            actuaDemonio();
        FinPlanifActiv;
    end
    devolver mejor solución
end
```

3. actuaDemonio: Las acciones del demonio. Las acciones del demonio son acciones opcionales (que no tienen un contrapunto natural) para implementar tareas desde una perspectiva global que no pueden llevar a cabo las hormigas por la perspectiva local que ofrecen. Ejemplos son observar la calidad de todas las soluciones generadas y depositar una nueva cantidad de feromona adicional sólo en las transiciones/componentes asociadas a algunas soluciones, o aplicar un procedimiento de búsqueda local a las soluciones generadas por las hormigas antes de actualizar los rastros de feromona.

En [31, 96, 100, 263] aparecen varias de sus aplicaciones a la solución de diferentes problemas, tales como problemas de optimización estática combinatoria, viajante de comercio, asignación cuadrática, planificación de horarios de trabajo, encaminamiento de vehículos, ordenamiento secuencial, coloración de grafos, etc.

2.5. Metaheurísticas híbridas

En la literatura [127,143,211] nos encontramos una gran variedad de problemas y métodos que aparecen dentro de la optimización heurística. La principal tendencia dentro de este campo es el desarrollo de nuevas metaheurísticas que proporcionen mejores soluciones. Una de las líneas de desarrollo son las metaheurísticaas híbridas que tratan de combinar metaheurísticas ya existentes con la intención de aprovecharse de lo mejor de cada una de ellas. Dentro de estas últimas destacan las metaheurísticas cooperativas en las que las distintas metaheurísticas cooperan intercambiando información para resolver una instancia de un problema. En el siguiente capítulo se analizan este tipo de metaheurísticas.

Capítulo 3

Metaheurísticas cooperativas

Muchos estudios han mostrado que las heurísticas y metaheurísticas son buenas herramientas para proporcionar soluciones razonablemente buenas (excelentes en algunos casos) usando un número moderado de recursos. Una breve mirada a la literatura reciente [127, 143, 211] revela la gran variedad de problemas y métodos que aparecen dentro del tema global de optimización heurística. La principal tendencia actual en este campo es la de desarrollar nuevas metaheurísticas que proporcionen mejores soluciones para problemas dados. Sin embargo una línea muy interesante de desarrollo son las metaheurísticas cooperativas que tratan de combinar metaheurísticas ya existentes formando sistemas híbridos en los que éstas cooperen de forma paralela para resolver un problema. Esta aproximación puede ser interesante por dos razones: primero, se pueden resolver instancias de problemas más grandes y segundo se pueden obtener herramientas robustas que ofrezcan soluciones de gran calidad a pesar de las variaciones en las características de las instancias.

Dos campos de las metaheurísticas tienen gran influencia en el desarrollo de metaheurísticas cooperativas: Las metaheurísticas paralelas y las metaheurísticas híbridas. A continuación se explicarán ambos campos en profundidad.

3.1. Metaheurísticas paralelas

El campo de las metaheurísticas paralelas aparece de manera natural en el desarrollo de las metaheurísticas como una alternativa para mejorar el factor de aceleración de la búsqueda de soluciones. Aunque posteriormente se ha podido constatar [7, 84] que pueden lograrse aplicaciones robustas que alcanzan soluciones de mayor calidad que sus correspondientes algoritmos secuenciales. La metaheurísticas paralelas se han aplicado a diversos campos, como: problemas de rutas [85], configuración de redes de telecomunicaciones [86], localización [124],

selección de características en ámbitos de biomedicina [205], etc.

3.1.1. Taxonomías de metaheurísticas paralelas

Los primeros esfuerzos por obtener metaheurísticas paralelas se centraron en la paralelización del temple simulado [1, 15, 134, 227]. Fruto de esta primera etapa aparecieron diversas clasificaciones de la paralelización de temple simulado, como la propuesta en [111] donde se distinguen dos tipos, paralelismo de un solo intento y paralelismo de múltiples intentos, pero la mayoría de métodos son dependientes del problema y sólo obtienen una mejora en velocidad significativa en los problemas para los que fueron desarrollados. Otra clasificación algo más ambiciosa aparece en [134], donde se distinguen tres tipos de paralelismo, algoritmos serial-like que mantienen las propiedades de los algoritmos secuenciales, algoritmos de generación alterada, que modifican la generación de estados para reducir comunicación pero retienen cálculos de costo exactos y algoritmos asíncronos que reducen la comunicación aun más calculando el costo usando información no actualizada.

El interés por paralelizar estrategias de optimización pronto se extendió a otras metaheurísticas, como la búsqueda tabú o los algoritmos genéticos. La primera clasificación de búsqueda tabú paralelos fue propuesta por Voss [252], basada en el uso de diferentes estrategias de búsqueda y soluciones iniciales. Después fue ampliamente generalizada por Crainic et al. [84, 90] teniendo en cuenta aspectos adicionales como organización de la comunicación y manejo de información. De igual manera para estrategias paralelas basadas en algoritmos genéticos aparecen clasificaciones como la propuesta en [68] que distinguen 4 tipos de algoritmos genéticos paralelos:

- Global single-population master-slave. Consisten en una única población para la cual la evaluación del fitness es distribuida.
- Fine-grained single-population. Consiste en una población espacialmente estructurada en la cual selección y cruce están restringidos a un pequeño vecindario.
- Multiple-population. Consisten en varias poblaciones que evolucionan independientemente, intercambiando ocasionalmente individuos, también conocida como modelo de isla.
- Hierarchical parallel algoritmos genéticos. A alto nivel son multiple-population GAs con single-population parallel algoritmos genéticos a nivel bajo.

Han aparecido algunos intentos de obtener clasificaciones más ambiciosas, que no fueran dependientes de la metaheurística. En este sentido en [91, 251] se proponen clasificaciones

similares para metaheurísticas basadas en trayectorias, que distinguen entre estrategias con una o varias trayectorias. En las estrategias de una sola trayectoria se suele paralelizar el análisis del vecindarios, mientras que en las de varias trayectorias cada una se ejecuta en un procesador puediendo ser cooperativas (en las que se produzca algún intercambio de información) o independientes.

Finalmente, en [89] se propuso una clasificación que pretende englobar todos los tipos de metaheurísticas y que ha acabado imponiéndose. Esta clasificación distingue entre 3 tipos de paralelismo:

- Paralelismo de Tipo 1 o de Bajo Nivel. Esta fuente de paralelismo se encuentra normalmente dentro de una iteración del método de búsqueda, y se puede obtener a través de ejecuciones concurrentes de las operaciones o de evaluaciones concurrentes de varios movimientos que llevan a cabo una iteración del método de búsqueda. Está orientado directamente a reducir el tiempo de ejecución del método y no a lograr mayor exploración o soluciones de mayor calidad; y conviene señalar que, dado el mismo número de iteraciones, tanto la implementación secuencial como en paralelo produce el mismo resultado.
- Paralelismo de Tipo 2 o Descomposición del Dominio. Se obtiene el paralelismo descomponiendo las variables de decisión en conjuntos disjuntos. Esta descomposición reduce el tamaño del espacio de búsqueda, pero necesita ser repetida para permitir la exploración completa del espacio de solución. Una heurística particular es aplicada a cada variable en el subconjunto que se trate, los otros se consideran fijos. Esta estrategia se implementa generalmente en un modelo maestro-esclavo, donde el proceso maestro divide las variables de decisión.
- Paralelismo del Tipo 3 o Búsqueda Múltiple. Se lleva a cabo con varias búsquedas concurrentes en el espacio de soluciones. Cada hilo concurrente puede o no ejecutar el mismo método, puede empezar por las mismas o diferentes soluciones iniciales, etc. Los hilos pueden comunicarse durante la búsqueda o sólo al final para identificar la mejor de todas las soluciones. La primera estrategia se denomina método cooperativo de búsqueda múltiple y la segunda es conocida como método de búsqueda independiente. La comunicación puede ser realizada de manera síncrona o asíncrona y puede ser manejada por eventos o ejecutada en momentos decididos dinámicamente o predeterminados de antemano. Esta estrategia se usa frecuentemente para llevar a cabo una mayor exploración del espacio de búsqueda. Varios estudios [87, 88] han mostrado que las técnicas multihilos proporcionan mejores soluciones que su correspondiente contraparte secuencial, incluso cuando el tiempo disponible de ejecución para cada hilo es menor

que el de la computación secuencial. Dichos estudios han mostrado también que la combinación de distintos hilos, implementando cada uno de ellos diferentes estrategias, incrementa la robustez de la búsqueda global en relación con las variaciones de las características de las instancias del problema. También indican que mientras la estrategia de búsquedas independientes es fácil de implementar y obtiene buenos resultados, éstos se pueden mejorar usando la estrategia cooperativa de búsqueda múltiple.

3.2. Metaheurísticas híbridas

3.2.1. Introducción

Tradicionalmente las metaheurísticas se han clasificado, [32, 78], como basadas en poblaciones o en trayectorias. Las metaheurísticas basadas en poblaciones son las que conducen la evolución en el espacio de búsqueda de conjuntos de soluciones, usualmente llamados poblaciones, con la intención de acercarse a la solución óptima con sus elementos. El aspecto fundamental de estas metaheurísticas consiste en la interacción entre los miembros de la población frente a las búsquedas que se guían por la información de soluciones individuales. Por otro lado, las metaheurísticas basadas en trayectorias son aquellas que establecen estrategias para recorrer el espacio de soluciones del problema transformando de forma iterativa soluciones de partida. Estas dos familias tienen propiedades complementarias: las metaheurísticas basadas en poblaciones permiten una mejor exploración del espacio de búsqueda mientras que las basadas en trayectorias permiten una mejor intensificación de la búsqueda en aquellas áreas que son más prometedoras. La hibridación de metaheurísticas trata de aprovechar las distintas ventajas que ofrece cada metaheurística de tal forma que obtiene algoritmos más robustos que obtienen mejores soluciones.

La motivación tras la hibridación de diferentes conceptos algorítmicos es normalmente obtener sistemas que funcionen mejor, explotando y uniendo ventajas de las estrategias individuales puras, es decir se cree que estos híbridos se benefician de sinergia. El número de aplicaciones reportadas de metaheurísticas híbridas se incrementa considerablemente y los eventos científicos dedicados documentan la popularidad, éxito e importancia de esta línea de investigación específica. De hecho, hoy parece que elegir un híbrido adecuado es determinante para alcanzar alto rendimiento resolviendo los problemas más difíciles.

De hecho, la idea de hibridar metaheurísticas no es nueva, sino que data de los orígenes de las propias metaheurísticas. Al comienzo, sin embargo, los híbridos no eran tan populares puesto que existían diversas comunidades de investigadores fuertemente separadas o incluso competidoras, quienes consideraban su clase favorita de metaheurísticas la "mejor en general".

Mayormente se debe al teorema "No free lunch" que esta situación afortunadamente cambió y la gente reconoció que no puede existir una estrategia de optimización general que sea globalmente mejor que ninguna otra. De hecho, para resolver el problema más eficientemente, se requiere casi siempre un algoritmo especializado que necesita estar compuesto de las partes adecuadas.

Existen muchas publicaciones que proponen taxonomías para metaheurísticas híbridas o subcategorías particulares [32, 81, 113, 226, 241]. En este trabajo nos centraremos en la propuesta en [226] y que mostramos en la Fig. 3.1.

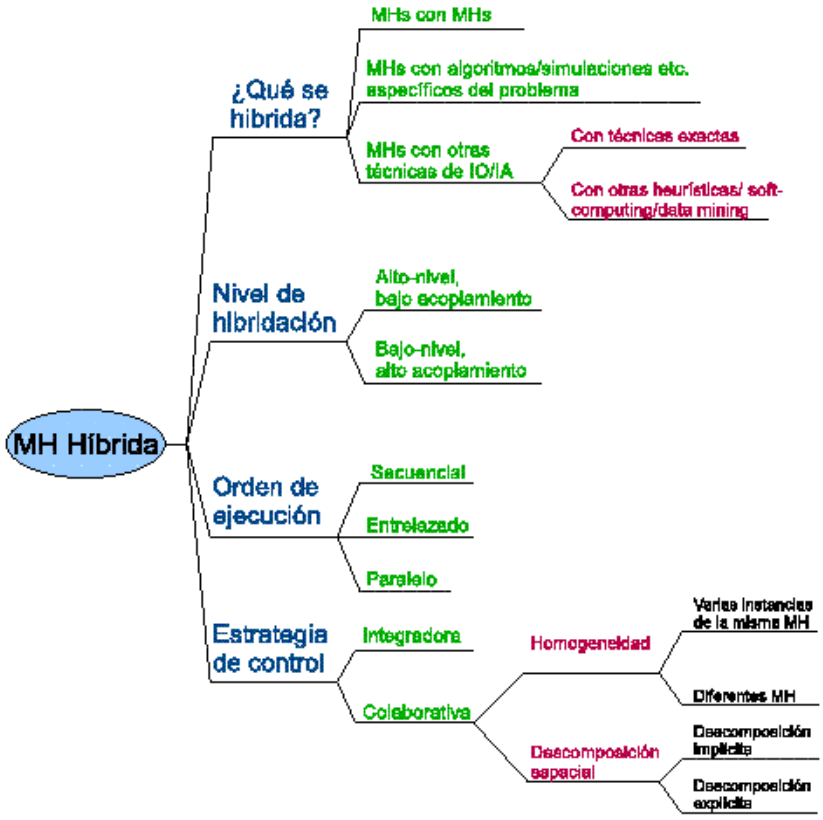


Figura 3.1: Taxonomía de metaheurísticas híbridas

Esta taxonomía combina aspectos de la propuesta por Talbi [241] con los puntos de vista de Cotta [81] y Blum et al. [32]. Con respecto a la hibridación de metaheurísticas con técnicas de optimización exactas usa conceptos de Puchinger y Raidl [222].

El primer nivel de distinción es *qué* se hibrida, es decir qué tipos de algoritmos. Se pueden combinar (a) distintas estrategias metaheurísticas, (b) metaheurísticas con ciertos algoritmos específicos del problema que se considere, tales como simulaciones especiales, o (c) metaheurísticas con otras técnicas más generales de investigación operativa (IO) como

ramificación y poda o programación lineal y de inteligencia artificial (IA) como técnicas de minería de datos o de lógica fuzzy.

Aparte de esta diferenciación, taxonomías previas de metaheurísticas híbridas [81, 241] distinguen principalmente el *nivel* (o fuerza) al que los diferentes algoritmos son combinados: las combinaciones de alto nivel retienen las identidades individuales de los algoritmos originales y cooperan sobre una interfaz relativamente bien definida; no hay relación directa y fuerte en el funcionamiento interno de los algoritmos. Por el contrario, los algoritmos de combinaciones de bajo nivel dependen fuertemente unos de otros, puesto que se intercambian componentes o funciones individuales.

Otra propiedad por la que se pueden distinguir sistemas híbridos es el *orden de ejecución*. En el modelo secuencial, un algoritmo es ejecutado estrictamente detrás del otro, y la información se pasa en un solo sentido. Un preproceso inteligente de los datos de entrada o un postproceso de los resultados de otro algoritmo caerían en esta categoría. Otro ejemplo son los problemas multi-nivel que son resueltos considerando un nivel después de otro usando algoritmos de optimización dedicados. Por el contrario, existen modelos entrelazados y paralelos, en los que los algoritmos pueden interaccionar en modos más sofisticados. Se pueden encontrar clasificaciones detalladas de metaheurísticas híbridas paralelas en [83, 113].

También se pueden distinguir metaheurísticas híbridas de acuerdo con su *estrategia de control*. Siguiendo [81, 222], existen combinaciones integradoras (coercitivas) y colaborativas (cooperativas). En aproximaciones integradoras, un algoritmo es considerado un subordinado, componente anidado del otro. Esta aproximación es muy popular, como ejemplo más claro se pueden considerar los algoritmos meméticos. En combinaciones colaborativas, los algoritmos intercambian información pero no son parte unos de otros. Por ejemplo, el modelo de isla para paralelizar algoritmos genéticos. Se puede clasificar en mayor profundidad diferenciando entre *homogéneos* y *heterogéneos*. Las estrategias homogéneas son aquellas en las que varias instancias de la misma metaheurística cooperan, como por ejemplo el modelo antes mencionado de isla. Por otro lado las estrategias heterogéneas combinan distintas metaheurísticas. Un ejemplo de ellas pueden ser los A-Teams propuestos en [242, 243], que son una arquitectura multiagente para resolver problemas de optimización que consisten en una colección de agentes y memorias, en la que los agentes trabajan asíncronamente y autónomamente sobre las memorias.

En particular en combinaciones colaborativas, otra cuestión es qué espacios de búsqueda son explorados por los algoritmos individuales. De acuerdo con [113] se puede distinguir entre una descomposición implícita resultante de soluciones iniciales diferentes, parámetros distintos, etc., y una descomposición explícita en la que cada algoritmo trabaje en un

subespacio definido explícitamente.

3.2.2. Hibridación de metaheurísticas con técnicas de minería de datos

Dentro de las metaheurísticas híbridas un campo que está cobrando cada vez mayor relevancia es la de combinar las metaheurísticas con técnicas de minería de datos. La minería de datos, es el proceso de explorar de modo automático grandes cantidades de datos para descubrir patrones. Para conseguir esto, la minería de datos usa técnicas computacionales de estadística, aprendizaje computacional, reconocimiento de patrones u optimización combinatoria. La combinación de minería de datos con el uso de heurísticas se ha convertido en un campo de investigación popular que ha mostrado resultados prometedores, en [161] se muestra una interesante revisión del campo.

La combinación de extracción de conocimiento con el uso de heurísticas y metaheurísticas es una línea muy interesante en optimización heurística. Ejemplos de este tipo de combinaciones son: [219], donde se presenta un sistema híbrido en el que un sistema inmune artificial y un mapa auto-organizativo se combinan (uno sirve para detectar anomalías en conexiones de red y el otro las categoriza usando un proceso de aprendizaje); [120, 121, 138, 215], donde se utilizan distintas técnicas de minería de datos con el objetivo de seleccionar el algoritmo que se utilizará para resolver una instancia de un problema; [43], donde una hiperheurística utiliza razonamiento basado en casos para decidir qué heurística se tiene que utilizar en cada momento; [244], donde se utiliza clasificación asociativa, en lugar de razonamiento basado en casos, con el mismo objetivo; o [19, 20, 112, 199, 221], en los que se proponen distintas técnicas para aprender los parámetros que debe utilizar una metaheurística.

Para ilustrar las diferentes maneras en las que se puede integrar el conocimiento en las metaheurísticas, en [161] se propuso una pequeña taxonomía que sumariza composiciones encontradas en muchos artículos, y que se puede ver en la Fig. 3.2.

- Se pueden distinguir dos tipos de conocimiento: conocimiento adquirido con anterioridad, llamado *conocimiento a priori* y conocimiento adquirido dinámicamente, *conocimiento dinámico*, que es extraído o descubierto durante la búsqueda.
- Otra información interesante para clasificar los algoritmos es distinguir el objetivo de la cooperación. La cooperación se puede utilizar o bien para reducir el tiempo de computación, es decir, técnicas de aceleración mediante la simplificación de la función de fitness (funciones de aproximación), o la reducción del espacio de búsqueda (guiando a la metaheurística a las regiones más prometedoras). O bien la cooperación puede ser usada para incrementar la calidad de la búsqueda, introduciendo conocimiento en

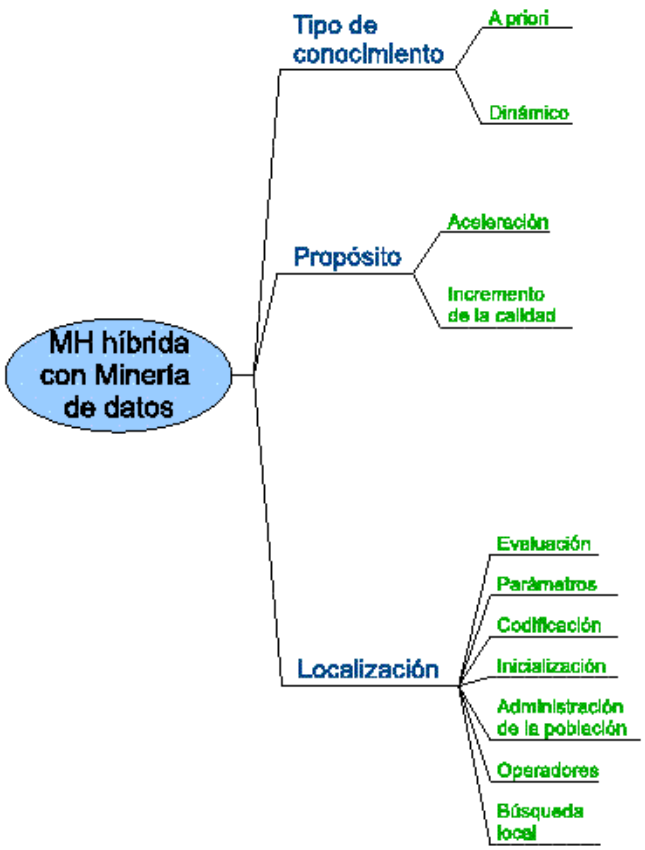


Figura 3.2: Taxonomía de metaheurísticas híbridas y minería de datos

operadores o en otras partes de la metaheurística. De hecho, la inserción de técnicas de minería de datos a menudo conlleva tanto la aceleración como el incremento de calidad.

- El último punto usado para distinguir los híbridos es determinar qué parte de la metaheurística se ve afectada por la incorporación del conocimiento. La hibridación puede ocurrir en cualquier parte de la metaheurística: parámetros, codificación, evaluación, inicialización, operadores, etc.

3.3. Metaheurísticas paralelas cooperativas

Este trabajo se enmarca en el paralelismo de tipo 3 o de búsqueda múltiple y más específicamente dentro de las estrategias cooperativas. Dentro de este campo se han centrado muchos esfuerzos, de manera que se pueden encontrar implementaciones síncronas donde la información es compartida en intervalos regulares como en el caso de [90] para la búsqueda tabú, [176] para el temple simulado o [9] para un algoritmo genético y una búsqueda dispersa. Más recien-

temente se han desarrollado implementaciones multi-búsqueda cooperativas asíncronas [88] y búsquedas cooperativas multinivel [17] que, de acuerdo con los informes en [89] dan mejores resultados que las implementaciones síncronas. Por último el amplio uso de metaheurísticas paralelas que se está produciendo ha llevado a que se desarrollen frameworks para su implementacion, como ParadisEO [65], MAGMA [192] o MALLBA [8] que además permiten el desarrollo de metaheurísticas híbridas.

Como se ha dicho antes, las metaheurísticas de búsqueda múltiple cooperativa obtienen soluciones de una mejor calidad que las soluciones obtenidas por métodos independientes. Pero estudios previos [87,88] demuestran que los métodos cooperativos con un acceso no restringido a la información compartida pueden experimentar problemas de convergencia prematura. Esto parece deberse a la estabilización de la información compartida que se produce como resultado del intenso intercambio de las mejores soluciones. Por ello sería interesante encontrar una manera de controlar este intercambio de información. Así surgen propuestas como la de Pelta en [214] donde se propone una estrategia cooperativa que se basa en la memoria para controlar este efecto. En ella hay un agente coordinador modelado por una serie de reglas fuzzy definidas de forma "ad hoc", que recibe información de un conjunto de agentes resolvedores y envía instrucciones a cada uno de ellos sobre cómo deben continuar. Cada agente resolvedor implementa la metaheurística FANS [30] como un clon.

Sin embargo, parece que una estrategia cooperativa basada en una única metaheurística no cubre todas las posibilidades y se recomienda el uso de estrategias que combinen diferentes metaheurísticas. Estudios en esta dirección los encontramos en [55,131,175].

3.4. Conclusiones

En este capítulo se ha hecho una revisión de los trabajos más relevantes en el campo de las metaheurísticas cooperativas. A través de esta revisión hemos sido capaces de observar que, a pesar de la cantidad de propuestas, todavía queda muchos campos por explorar. En particular, se ha observado un escaso estudio en el caso de metaheurísticas cooperativas heterogeneas, que sin embargo son consideradas las más prometedoras. Dentro de estas se ha comprobado que no existen aproximaciones que sean capaces de abordar los problemas utilizando conocimiento previo acerca del mismo, por lo que se podría decir que actúan a ciegas. Por ello, en esta tesis nos centraremos en el desarrollo de metaheurísticas cooperativas cuyo control y configuración sea realizado a través del conocimiento previo que se pueda poseer del problema a resolver, en particular, de conocimiento previo obtenido a partir de un proceso de extracción inteligente del conocimiento.

Parte II

UN MARCO GENERAL PARA EL DISEÑO DE METAHEURÍSTICAS COOPERATIVAS HETEROGENEAS CON CONOCIMIENTO A PRIORI

Capítulo 4

Elementos básicos

4.1. Introducción

En este capítulo introducimos brevemente algunos de los elementos básicos necesarios para la construcción de la estrategia cooperativa. Para más detalle de estos elementos pueden leerse los apendices A, B y C, correspondientes.

4.2. Elementos básicos de SoftComputing

Desde que Zadeh con su trabajo seminal, [266], introdujo el concepto de conjunto fuzzy y consecuentemente se extendió al concepto de variables lingüísticas, [267–269], la popularidad y el uso de la lógica fuzzy han sido extraordinarios. En 1994, L.A. Zadeh, [270], dio la primera definición de "Soft Computing", y hasta entonces las referencias a los conceptos que actualmente ésta maneja, solían hacerse de manera aislada con indicación del empleo de metodologías fuzzy. Zadeh propuso la definición de Soft Computing, estableciéndola en los siguientes términos:

> "Básicamente, Soft Computing no es un cuerpo homogéneo de conceptos y técnicas. Mas bien es una mezcla de distintos métodos que de una forma u otra cooperan desde sus fundamentos. En este sentido, el principal objetivo de la Soft Computing es aprovechar la tolerancia que conllevan la imprecisión y la incertidumbre, para conseguir manejabilidad, robustez y soluciones de bajo costo. Los principales ingredientes de la Soft Computing son la Lógica Fuzzy, la Neurocomputación y el Razonamiento Probabilístico, incluyendo este último a los Algoritmos Genéticos, las Redes de Creencia, los Sistemas Caóticos y algunas partes de la Teoría de Aprendizaje. En esa asociación de Lógica Fuzzy, Neurocomputación y Razonamiento Probabilístico, la Lógica Fuzzy se ocupa principalmente de la imprecisión y el Razonamiento Aproximado; la Neurocomputación del aprendiza-

> je, y el Razonamiento Probabilístico de la incertidumbre y la propagación de las creencias".

Queda claro que la Soft Computing no está definida precisamente, sino que en una primera aproximación se define por extensión, por medio de distintos conceptos y técnicas que intentan superar las dificultades que surgen en los problemas reales que se dan en un mundo que es impreciso, incierto y difícil de categorizar.

A partir de ella, se han realizado varios intentos de ajustarla. Verdegay, Yager y Bonissone han presentado en [250] una definición más precisa e ilustrativa de lo que es la Soft Computing en la actualidad, en los siguientes términos:

> "El punto de vista que aquí consideramos es otra forma de definir la Soft Computing, por medio de la cual se la considera como la antítesis de lo que podríamos llamar Hard Computing. Este punto es consistente con el presentado en [270, 271]. La Soft Computing puede, por tanto, verse como un conjunto de técnicas y métodos que permitan tratar las situaciones prácticas reales de la misma forma que suelen hacerlo los seres humanos, es decir, en base a inteligencia, sentido común, consideración de analogías, aproximaciones, etc. En este sentido Soft Computing es una familia de métodos de resolución de problemas cuyos primeros miembros serían el Razonamiento Aproximado y los Métodos de Aproximación Funcional y de Optimización, incluyendo los de búsqueda. En este sentido, la Soft Computing queda situada como la base teórica del área de los Sistemas Inteligentes.".

En esta tesis nos movemos en las dos grandes áreas del Soft Computing: el Razonamiento Aproximado y la Aproximación Funcional/Métodos de Optimización. En la memoria utilizaremos los elementos básicos de la lógica fuzzy, conjuntos fuzzy, etc, por ello, en esta sección se describen los elementos básicos principales necesarios para el desarrollo de esta parte de la memoria.

4.2.1. Lógica y Conjuntos Fuzzy

La Lógica Fuzzy se plantea como alternativa a la lógica tradicional con el objetivo de introducir grados de incertidumbre en las sentencias que califica, [272]. Hay situaciones en las que la lógica tradicional funciona perfectamente, sin embargo, el inconveniente de esta lógica es que en la vida real no nos encontramos frecuentemente con criterios de clasificación nítidos. Normalmente, la información no puede ser evaluada cuantitativamente de forma precisa, pero puede que sí sea posible hacerlo cualitativamente, y en estos casos hemos de

hacer uso de un *enfoque lingüístico*. Por ejemplo, cuando intentamos cualificar algún fenómeno relacionado con percepciones humanas, a menudo usamos palabras o descripciones en lenguaje natural, en lugar de valores numéricos. Supongamos que dado un conjunto de personas, las intentamos agrupar según su altura. Las personas no son simplemente *altas* o *bajas* sino que la mayoría pertenecen a grupos de altura intermedia. La gente suele ser *más bien alta* o *de altura media*. Casi nunca las calificamos con rotundidad, porque el lenguaje que usamos nos permite introducir modificadores que añaden imprecisión: *un poco, mucho, algo, ...*

Como la lógica tradicional es bivaluada (sólo admite dos valores: o el elemento pertenece al conjunto o no pertenece), se ve incapacitada para agrupar según su altura al anterior conjunto de personas, puesto que su solución sería definir un umbral de pertenencia (por ejemplo, un valor que todo el mundo considera que, de ser alcanzado o superado, la persona en cuestión puede llamarse *alta*). Si dicho umbral es 1.80, todas las personas que midan 1.80 o más serán *altas*, mientras que el resto serán bajas. Según esta manera de pensar, alguien que mida 1.79 será tratado igual que otro que mida 1.50, ya que ambos han merecido el calificativo de personas *bajas*.

Si dispusiéramos de una herramienta para caracterizar las alturas de forma que las transiciones entre las que son altas y las que no lo son fueran suaves, estaríamos reproduciendo la realidad mucho más fielmente. En la realidad hay unos puntos de cruce donde las personas dejan de ser *altas* para ser consideradas *medianas*, de forma que el concepto de *alto* decrece linealmente con la altura. Asignando una función lineal para caracterizar el concepto *alto* en lugar de definir un sólo umbral de separación estamos dando mucha más información acerca de los elementos. Esta función, como veremos, se llamará función de pertenencia.

En este sentido, el uso de la Teoría de Conjuntos Fuzzy ha dado muy buenos resultados para el tratamiento de información de forma cualitativa, [267–269]. El *modelado lingüístico fuzzy* es una herramienta que permite representar aspectos cualitativos y que está basada en el concepto de *variables lingüísticas*, es decir, variables cuyos valores no son números, sino palabras o sentencias expresadas en lenguaje natural o artificial, [267–269]. Cada valor lingüístico se caracteriza por un valor sintáctico o *etiqueta* y un valor semántico o *significado*. La etiqueta es una palabra o sentencia perteneciente a un conjunto de términos lingüísticos y el significado es un subconjunto fuzzy en un universo de discurso.

Se ha demostrado que es una herramienta muy útil en numerosos problemas, como por ejemplo en *recuperación de información*, *evaluación de servicios*, *toma de decisiones*, *procesos de consenso*, etc, [13, 24, 25, 151, 152, 183].

4.2.2. Conjuntos Fuzzy y Funciones de Pertenencia

La noción de conjunto refleja la tendencia a organizar, generalizar y clasificar el conocimiento sobre los objetos del mundo real. El encapsulamiento de los objetos es una colección cuyos miembros comparten una serie de características o propiedades que implican la noción de conjunto. Los conjuntos introducen una noción de dicotomía, que en esencia es una clasificación binaria: o se acepta o se rechaza la pertenencia de un objeto a una categoría determinada. Habitualmente la decisión de *aceptar* se denota como 1 y la de *rechazar* como 0. Esta decisión de aceptar o rechazar se expresa mediante una función característica, según las propiedades que posean los objetos del conjunto. Sea U un conjunto cuyos elementos denotaremos por x, y sea A un subconjunto de U. La pertenencia de un elemento x de U al conjunto A viene dada por la función característica

$$\mu_A(x) = \begin{cases} 1 & \text{si y sólo si } x \in A \\ 0 & \text{si y sólo si } x \notin A \end{cases}$$

$\{0,1\}$ es el llamado conjunto valoración.

La Lógica Fuzzy se fundamenta en el concepto de *conjunto fuzzy*, [266], que suaviza el requerimiento anterior y admite valores intermedios en la función característica, que se denomina *función de pertenencia*. Esto permite una interpretación más realista de la información, puesto que la mayoría de las categorías que describen los objetos del mundo real, no tienen unos límites claros y bien definidos.

Un conjunto fuzzy puede definirse como una colección de objetos con valores de pertenencia entre 0 (exclusión total) y 1 (pertenencia total). Los valores de pertenencia expresan los grados con los que cada objeto es compatible con las propiedades o características distintivas de la colección. Formalmente podemos definir un conjunto fuzzy como sigue.

Definición 4.1. Un *conjunto fuzzy* $\widetilde{A}$ sobre un dominio o universo de discurso U está caracterizado por una función de pertenencia que asocia a cada elemento del conjunto el grado con que pertenece a dicho conjunto, asignándole un valor en el intervalo [0,1], $\mu_{\widetilde{A}} : U \rightarrow [0,1]$

□

Así, un conjunto fuzzy $\widetilde{A}$ sobre U puede representarse como un conjunto de pares ordenados de un elemento perteneciente a U y su grado de pertenencia, $\widetilde{A} = \{(x, \mu_{\widetilde{A}}(x)) \ / \ x \in U, \mu_{\widetilde{A}}(x) \in [0,1]\}$. Por ejemplo, consideremos el concepto *persona alta*, en un contexto donde la estatura oscila entre 1 y 2 m. Como es de suponer, alguien que mida 1,30 m. no se puede considerar como *persona alta* por lo que su grado de pertenencia al conjunto de personas altas será de 0. Por el contrario, una persona que mida 1,90 m. sí la consideramos alta por lo que su grado de pertenencia al conjunto es de 1.

Las gráficas que representan una función de pertenencia pueden adoptar cualquier forma, cumpliendo propiedades específicas, pero es el contexto de la aplicación lo que determina la representación más adecuada en cada caso. Puesto que las valoraciones lingüísticas dadas por los usuarios son únicamente aproximaciones, algunos autores consideran que las funciones de pertenencia trapezoidales lineales son suficientemente buenas para capturar la imprecisión de tales valoraciones lingüísticas. La representación paramétrica es obtenida a partir de una 4-tupla (r, a, b, R), donde a y b indican el intervalo en que el valor de pertenencia es 1, con r y R indicando los límites izquierdo y derecho del dominio de definición de la función de pertenencia trapezoidal. Un caso particular de este tipo de representación son las valoraciones lingüísticas cuyas funciones de pertenencia son triangulares, es decir, $a = b$, por lo que se representan por medio de una 3-tupla (r, a, R). La figura 4.1 muestra la descripción y la representación gráfica de un ejemplo de función de pertenencia trapezoidal.

$$\mu_{\widetilde{A}}(x, r, a, b, R) = \begin{cases} 0 & x < r \\ \frac{x - r}{a - r} & r \leq x \leq a \\ 1 & a \leq x \leq b \\ \frac{R - x}{R - b} & b \leq x \leq R \\ 0 & x > R \end{cases}$$

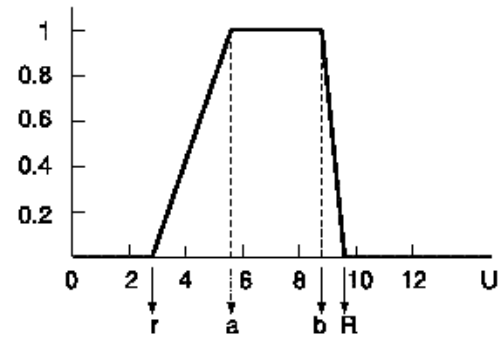


Figura 4.1: Ejemplo de función de pertenencia.

4.2.2.1. Definiciones Básicas

Definición 4.2. Se define *soporte* de un conjunto fuzzy $\widetilde{A}$ en el universo U, como el conjunto formado por todos los elementos de U cuyo grado de pertenencia a $\widetilde{A}$ sea mayor que 0:

$$supp(\widetilde{A}) = \{x \in U \ / \ \mu_{\widetilde{A}}(x) > 0\}$$

□

Definición 4.3. El *α-corte* de un conjunto fuzzy $\widetilde{A}$ es el conjunto formado por todos los elementos del universo U cuyos grados de pertenencia en $\widetilde{A}$ son mayores o iguales que el valor de corte $\alpha \in [0, 1]$:

$$A_\alpha = \{x \in U \ / \ \mu_{\widetilde{A}}(x) \geq \alpha\}$$

□

Definición 4.4. Se denomina *conjunto de niveles* de un conjunto fuzzy $\widetilde{A}$, al conjunto de grados de pertenencia de sus elementos:

$$L(\widetilde{A}) = \{a \ / \ \mu_{\widetilde{A}}(x) = a, \ \ x \in U\}$$

□

4.3. Elementos básicos de Agentes y Sistemas Multiagente

4.3.1. Concepto general de Agente

Los primeros artículos que contienen la idea de agente software aparecen alrededor de 1980, en los cuales se plantea la cooperación entre sistemas expertos para la resolución de problemas complejos. Estos sistemas expertos se diseñan como aplicaciones basadas en conocimiento que permiten resolver problemas en dominios específicos a través de procesos de razonamiento [144]. Sin embargo, pronto se detecta que su capacidad de resolución es muy limitada y que se necesita intercambiar conocimiento entre ellos para paliar estas limitaciones. Surge entonces la idea de tomar cada sistema experto como un *agente de colaboración* con otros sistemas con el fin de aumentar su capacidad individual de resolución de problemas. A partir de ese momento, la tecnología de agente ha ido avanzando progresivamente mediante la adopción de técnicas de diversos campos tales como Inteligencia Artificial, Sistemas Distribuidos e Ingeniería del Software.

Junto a esta continua evolución han ido apareciendo múltiples definiciones de agente software. Las definiciones más utilizadas para describir el estado actual de esta tecnología pueden encontrarse en los artículos de Wooldridge [261] y Luck [182], donde se recogen los siguientes aspectos:

> Un agente software es un programa computacional que actúa en representación de una *entidad* y en un determinado *entorno* para conseguir ciertos *objetivos* de una forma *autónoma* e *inteligente*, y en la que posiblemente necesite *interaccionar* con otros agentes.

4.3.2. Tipos de Agentes

Los criterios para clasificar agentes vienen dados por los diferentes ángulos desde los que pueden ser observados. Algunas propuestas para la definición de criterios pueden consultarse en [35, 204, 261]. Los criterios considerados más importantes para un agente son tres: su comportamiento individual, el modo de interacción con otros agentes y su utilidad.

a) Tipos de agentes según su comportamiento individual

Este criterio clasifica a los agentes según la manera de actuar a partir de la información recibida de su entorno.

Agentes reactivos [33, 94]. El modelo de computación de estos agentes está basado en el ciclo *recepción de eventos/reacción*. La recepción de eventos está gestionada por el módulo de percepción, mientras que la reacción se divide en dos fases. En la primera, el módulo de control selecciona un procedimiento sencillo para reaccionar ante el evento recibido teniendo en cuenta el estado interno del agente. En la segunda, este procedimiento es ejecutado por el módulo de acción, dando lugar a cambios en dicho estado interno o a acciones sobre el entorno. El siguiente párrafo muestra cómo funciona el módulo de control de estos agentes.

La arquitectura típica del módulo de control de los agentes reactivos puede observarse en la Figura 4.2. Esta arquitectura está compuesta por varios módulos de comportamiento

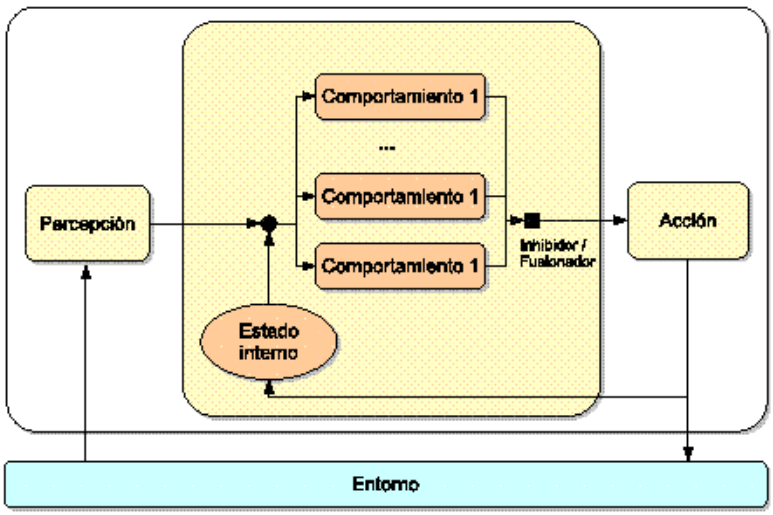


Figura 4.2: Arquitectura del módulo de control en agente reactivos.

donde cada uno contiene un procedimiento básico para resolver una tarea específica. Estos comportamientos son activados según el evento recibido y el estado interno del agente, mediante el uso de autómatas de estados finitos o redes neuronales. En caso de activarse más de un módulo de comportamiento es necesario definir un conjunto de relaciones entre ellos para decidir cuál es el más adecuado según la situación, inhibiendo el resto de comportamientos. Otra posibilidad es fusionar comportamientos mediante una ponderación de éstos. En cualquier caso, observar que un agente reactivo no tiene mecanismos explícitos para representar conocimiento ni utiliza procesos de razonamiento sobre la información recibida cuando selecciona un comportamiento, por lo que está diseñado para resolver problemas sencillos. La

inteligencia en estos agentes no es un propiedad individual de cada uno, sino que emerge de la interacción entre ellos cuando se comportan como una comunidad.

Las ventajas de los agentes reactivos radican en la sencillez para definir comportamientos y en sus respuestas prácticamente instantáneas ante la llegada de eventos. Por tanto, su aplicación está orientada a entornos impredecibles y altamente cambiantes donde el agente tiene que dar una respuesta rápida y eficaz. De hecho, este tipo de agentes ha sido aplicado principalmente en robótica. En cambio, tanto la falta de un modelo para representar conocimiento como la ausencia de procesos de razonamiento desaconsejan su uso en problemas donde el agente necesita procesar la información recibida y planificar las acciones que se tienen que llevar a cabo.

Agentes racionales [190,228]. También llamados *cognitivos*, su modelo de computación se basa en percibir información, integrarla en un modelo de conocimiento, razonar sobre dicho modelo y actuar según este razonamiento. Así, estos agentes utilizan algún tipo de modelo simbólico para representar conocimiento. Tanto la información recibida del entorno como la propia información que posee el agente sobre el problema que está resolviendo se integran en el modelo de conocimiento. Para alcanzar sus objetivos, los agente racionales emplean procesos de razonamiento sobre el modelo de conocimiento, y en algunos casos utilizan procesos de planificación y/o aprendizaje. Como resultado de estos procesos se obtienen las acciones que debe realizar el agente acompañadas siempre de una justificación. Estas acciones puede modificar los objetivos del agente e incidir sobre el entorno. El módulo de control que gestiona los procesos de integración de conocimiento y razonamiento está diseñado mediante una arquitectura deliberativa, la cual puede observarse en la Figura 4.3.

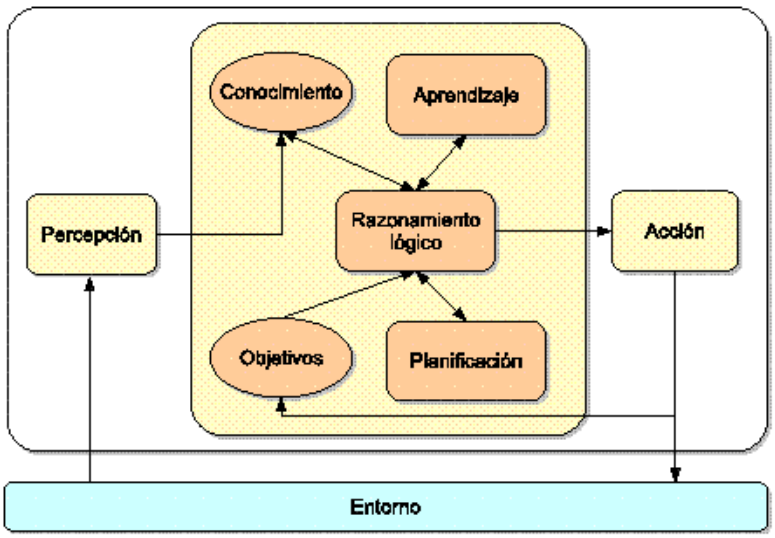


Figura 4.3: Arquitectura del módulo de control en agente racionales.

Una de las piezas fundamentales de esta arquitectura es el conocimiento del agente, representado a través de un modelo simbólico como ya hemos comentado. Dicho modelo se obtiene mediante formalismos computacionales tales como esquemas conceptuales combinados con programas de reglas. A su vez, los formalismos utilizados suelen estar basados en diferentes lógicas: lógica de predicados, lógica modal, lógica temporal, etc. La Figura 4.3 muestra también otro componente básico de la arquitectura: el módulo de razonamiento lógico. Este modulo toma el conocimiento y los objetivos del agente como entrada para obtener las acciones que hay que llevar a cabo. Además, el proceso de razonamiento puede estar apoyado por técnicas de planificación y aprendizaje. El resultado obtenido suele contener un plan con las acciones a ejecutar.

b) Tipos de agentes según su modo de interacción

El criterio de interacción entre agentes permite clasificar dichos agentes como entidades individuales que tienen un rol dentro de una organización. Así, podemos hacer una primera distinción entre agentes **individualistas** y agentes **cooperativos**. Los primeros son también conocidos como agentes *egoístas* que buscan conseguir sus objetivos en el sistema compitiendo con otros agentes y normalmente actúan por sí solos. En cambio, los agentes cooperativos colaboran con otros agentes compartiendo recursos y normalmente forman parte de una organización dentro del SMA en la que comparten un fin común.

Centrándonos en la dinámica del rol que los agentes toman en la organización podemos hacer una segunda clasificación de agentes, ya sean individualistas o cooperativos. La mayoría de ellos toman un rol *estático*, realizando un conjunto de tareas fijas y cuyas relaciones con el resto de agentes de la organización son inmutables (e.g., relaciones jerárquicas, cliente-servidor, etc.). Existen otros agentes con un rol *flexible* o *evolutivo*, los cuales pueden cambiar de objetivos, y en consecuencia cambiar el tipo de relaciones con otros agentes (e.g., agentes que pueden tomar varios roles en una organización, agentes que actúan como ofertantes y contratistas a la vez, etc.).

c) Utilidad del agente

Este criterio permite clasificar agentes según el propósito con el que fueron creados. Los factores que determinan la utilidad de un agente vienen dados por el dominio de aplicación y por el tipo de tareas que se va a realizar en tal dominio. Veamos estos factores.

En primer lugar, las áreas donde se ha aplicado el paradigma de agente software han ido creciendo progresivamente: comercio electrónico, telecomunicaciones, gestión de procesos de administración, gestión de conocimiento, etc. En segundo lugar, las tareas de los agentes en estos dominios van desde la monitorización de sistemas hasta la elaboración de diagnósticos,

pasando por la búsqueda de información, tareas de mediación y tareas de coordinación y resolución de conflictos entre agentes.

4.3.3. Sistemas Multiagente y comunicación en Sistemas Multiagente

4.3.3.1. Introducción

Como hemos visto, la idea original del concepto de agente está basada en representar entidades autónomas de forma que éstas puedan interaccionar entre ellas para la gestión inteligente de problemas complejos que no pueden ser abordados individualmente. Esta idea ha dado lugar a la construcción de sistemas multiagente (SMA, o *multiagent systems* (MAS) en inglés) [160, 256, 261]. El objetivo de estos sistemas es coordinar los distintos agentes que lo componen e integrar sus objetivos particulares en un objetivo común. Las aplicaciones de SMAs pueden abarcar diversos casos: la gestión de problemas cuyos elementos están distribuidos físicamente, cuando la solución para resolver un problema consiste en emplear un conocimiento heterogéneo mantenido por diferentes agentes, cuando cada agente ofrece una solución distinta al mismo problema y es necesario comparar y coordinar dichas soluciones para encontrar cuál es la mejor, etc.

Los sistema multiagente ofrecen un método par evitar las situaciones problemáticas descritas. En un sistema multiagente están activos diversos agentes autónomos independientes [41]. Cada uno de estos agentes se dedica a sus propios objetivos y sólo contacta con los otros agentes para obtener información, o para contribuir a una solución coordinada de un problema general. En ambas situaciones, cada agente individual tiene una tarea específica para la cual es adecuado y cuya solución no excede de sus capacidades. Esto permite el procesamiento de problemas complejos.

Los sistemas multiagente proporcionan una gran ventaja: permiten la integración de agentes existentes en un gran sistema. Por tanto, la solución de un problema no requiere el diseño y desarrollo de un nuevo agente especializado, en su lugar, puede ser utilizado el conocimiento de los agentes existentes combinándolo en un sistema multiagente o permitiendo que trabajen conjuntamente para resolver el problema.

4.3.3.2. Comunicación en Sistemas Multiagente

El problema que surge cuando dos seres inteligentes intentan interactuar es que necesita unas normas y un canal de comunicación: deben utilizar el mismo lenguaje, estar de acuerdo en el significado de los símbolos de ese lenguaje, tener un mecanismo de comunicación para

el intercambio de mensajes, no hablar al mismo tiempo, etc. En resumen, debe de existir comunicación y coordinación entre ellos.

Coordinación entre agentes

Cuando el número de agentes del sistema crece, aparece la necesidad de ayuda para localizar otros agentes que tengan la información o que nos puedan ofrecer los servicios que requerimos [207]. Muchos sistema multiagente utilizan agentes mediadores o facilitadotes que proporcionan determinados servicios y asistencia a los agentes del sistema (matchmaker, broker, blackboard, etc.). En general podemos distinguir tres categorías de agentes: P-agentes o agentes proveedores, R-agentes o agentes requeridores y agentes mediadores o intermediarios.

Métodos de comunicación

Existen diferentes métodos de comunicación. La invocación del procedimiento de un agente por otro agente representa el caso más simple. No obstante, sólo métodos de comunicación simples pueden implementarse utilizando llamadas a procedimientos. Por esta razón, las llamadas a procedimientos no se consideran un método de comunicación.

Los métodos de comunicación se pueden diferenciar en sistemas de pizarra y sistemas de mensaje/diálogo.

- **Sistemas de pizarra**

  La arquitectura de pizarra es ampliamente utilizada en Inteligencia Artificial. En un sistema con múltiples fuentes de conocimiento (o agentes independientes) necesitamos un mecanismo de comunicación. La pizarra ("blackboard") es una estructura de datos que es usada como un mecanismo general de comunicación entre las múltiples fuentes de conocimiento y es gestionada y arbitrada por un controlador [99, 202, 203]. Así, será un área de trabajo común a los agentes donde poder intercambiar información, datos y conocimiento.

  Como cada agente trabaja con su parte del problema, acudirá a la pizarra para ver nueva información puesta por otros agentes y, a su vez, pondrá sus resultados. La forma de funcionar será: un agente inicia una comunicación escribiendo algún tipo de información en la pizarra; en ese momento, estos datos están disponibles para todos los agentes del sistema; cada agente accederá a la pizarra eventualmente para comprobar si hay información nueva; normalmente cada agente no recogerá toda la información que se vaya escribiendo en la pizarra, sino que sólo obtendrá aquella que le interese, tal vez por pertenecer a conocimiento afín. La figura 4.4 ilustra la estructura de un sistema de pizarra.

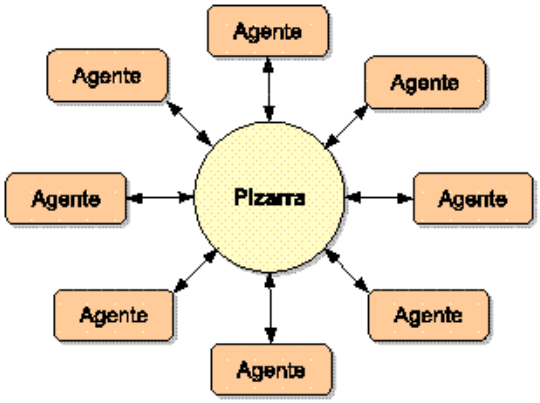


Figura 4.4: Estructura de un sistema de pizarra

Como se puede observar no hay comunicación directa entre agentes; por esto mismo, cada agente se ve obligado a resolver de forma autónoma su subproblema, bajo su responsabilidad. En este tipo básico de pizarras no existen áreas privadas; los agentes pueden escribir la información que deseen, y pueden acceder a toda la información contenida en la pizarra. Por tanto, si hay un número grande de agentes, la información contenida en la pizarra crece exponencialmente; y por otra parte, los agentes deberán buscar en una gran cantidad de información por cada acceso que realicen a la pizarra. Para optimizar este proceso, existen métodos más complejos (ampliaciones del original) para definir regiones en la pizarra, de manera que un agente sólo ve la región de la pizarra que tenga asignada. Un sistema multiagente puede disponer de múltiples pizarras.

- **Sistemas de mensajes**

Para desarrollar este tipo de interacción entre agentes existen tres elementos fundamentales a tener en cuenta. El primero es un *lenguaje de comunicación* para definir la estructura de los mensajes y sus actos comunicativos asociados, i.e. las acciones que representan (e.g., una orden, una propuesta, un paso de información, etc.). El segundo es una tecnología para representar el *contenido* del mensaje, i.e. la proposición que acompaña a la acción (e.g., la tarea a realizar o una pieza de información concreta). El tercer elemento es el *protocolo de interacción*, i.e. el conjunto de patrones de secuencia de mensajes que permite construir conversaciones entre agentes.

4.4. El proceso de extracción inteligente de conocimiento a partir de datos.

4.4.1. Introducción

Las bases de datos actuales han acumulado una gran variedad y cantidad de datos, estadísticas, índices, etc. en los cuales la información útil no es fácil de encontrar o inferir a simple vista. Con semejante cantidad de datos (y que cada vez crece más), el descubrimiento de conocimiento en bases de datos es una necesidad.

El proceso de extracción inteligente de conocimiento a partir de datos, también conocido como descubrimiento de conocimiento en bases de datos o KDD, se puede definir como el proceso no trivial de identificar patrones válidos, novedosos, potencialmente útiles y en última instancia, comprensibles a partir de grandes cantidades de datos que pueden estar en diferentes formatos. En esta definición se pueden ver los dos retos de este proceso: por un lado, trabajar con grandes volúmenes de datos, con los problemas que esto puede conllevar (ruido, datos ausentes, intratabilidad, volatilidad de los datos...), y por el otro utilizar técnicas adecuadas para analizar los mismos y extraer conocimiento novedoso y útil.

Así, los sistemas de extracción inteligente de conocimiento permiten la selección, limpieza, transformación y proyección de los datos; analizar los datos para extraer patrones y modelos adecuados; evaluar e interpretar los patrones para convertirlos en conocimiento; consolidar el conocimiento resolviendo posibles conflictos con conocimiento previamente extraído; y hacer el conocimiento disponible para su uso.

4.4.2. Fases del proceso de extracción de conocimiento

El proceso de extracción inteligente de cononcimiento es un proceso iterativo e interactivo. Es iterativo ya que la salida de alguna de las fases puede hacer volver a pasos anteriores y porque a menudo son necesarias varias iteraciones para extraer conocimiento de alta calidad. Es interactivo, porque el usuario, o más generalmente un experto en el dominio del problema, debe ayudar en la preparación de los datos, validación del conocimiento extraído, etc.

Este proceso se organiza en torno a cinco fases, como se ilustra en la figura 4.5. En la **fase de integración y recopilación de los datos** se determinan las fuentes de información que pueden ser útiles y dónde conseguirlas. A continuación, se transforman todos los datos a un formato común, detectando y resolviendo las inconsistencias. Dado que los datos provienen de diferentes fuentes, pueden contener valores erróneos o faltantes. Estas situaciones se tratan en la **fase de selección, limpieza y transformación**, en la que se eliminan o corrigen los datos

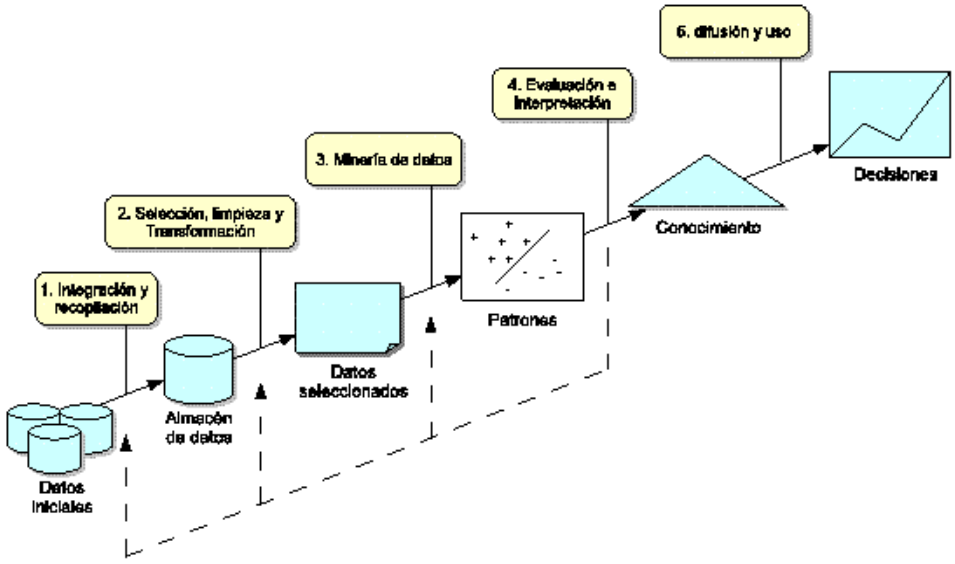


Figura 4.5: Fases del proceso de extracción de conocimiento.

incorrectos y se decide la estrategia a seguir con los datos incompletos. Además, se proyectan los datos para considerar únicamente aquellas variables o atributos que van a ser relevantes, con el objetivo de hacer más fácil la tarea propia de minería y para que los resultados de la misma sean más útiles. La selección incluye tanto una criba o fusión horizontal (filas / registros) como vertical (columnas / atributos). Las dos primeras fases se suelen englobar bajo el nombre de "preparación de datos". En la **fase de minería de datos**, se decide cuál es la tarea a realizar (clasificar, agrupar, etc.) y se elige el método que se va a utilizar. En la **fase de evaluación e interpretación** se evalúan los patrones y se analizan por los expertos, y si es necesario se vuelve a las fases anteriores para una nueva iteración. Esto incluye resolver posibles conflictos con el conocimiento que se disponía anteriormente. Finalmente, en la **fase de difusión** se hace uso del nuevo conocimiento y se hace partícipe de él a todos los posibles usuarios. A continuación, describimos las fases 2, 3 y 4 por ser el núcleo de estos procesos.

4.4.2.1. Fase de selección, limpieza y transformación

La calidad del conocimiento descubierto no sólo depende del algoritmo de minería utilizado, sino también de la calidad de los datos minados. Por ello, después de la recopilación, el siguiente paso en el proceso de extracción de conocimiento es seleccionar y preparar el subconjunto de datos que se va a minar, los cuales constituyen lo que se conoce como *vista minable*. Este paso es necesario ya que algunos datos coleccionados en la etapa anterior son irrelevantes o innecesarios para la tarea de minería que se desea realizar.

Pero además de la irrelevancia, existen otros problemas que afectan a la calidad de los datos. Uno de estos problemas es la presencia de valores que no se ajustan al comportamiento general de los datos (outliers). Estos datos anómalos pueden representar errores en los datos o pueden ser valores correctos que son simplemente diferentes a los demás. Algunos algoritmos de minería de datos ignoran estos datos, otros los descartan considerándolos ruido o excepciones, pero otros son muy sensibles y el resultado se ve claramente perjudicado por ello. Sin embargo, no siempre es conveniente eliminarlos, ya que, en algunas aplicaciones como la detección de compras fraudulentas efectuadas con tarjetas de crédito o la predicción de inundaciones, los eventos raros pueden ser más interesantes que los regulares.

La presencia de datos faltantes o perdidos (missing values) puede ser también un problema pernicioso que puede conducir a resultados poco precisos. No obstante, es necesario reflexionar primero sobre el significado de los valores faltantes antes de tomar ninguna decisión sobre cómo tratarlos ya que éstos pueden deberse a causas muy diversas, como a un mal funcionamiento del dispositivo que hizo la lectura del valor, a cambios efectuados en los procedimientos usados durante la colección de los datos o al hecho de que los datos se recopilen desde fuentes diversas.

Estos dos problemas son sólo dos ejemplos que muestran la necesidad de la limpieza de datos, es decir, de mejorar su calidad. Pero no es sólo suficiente con tener una buena calidad de datos, sino además poder proporcionar a los métodos de minería de datos el subconjunto de datos más adecuado para resolver el problema. Para ello es necesario seleccionar los datos apropiados.

La selección de atributos relevantes es uno de los preprocesamientos más importantes, ya que es crucial que los atributos utilizados sean relevantes para la tarea de minería de datos. Idealmente, se podrían usar todas las variables, pero es más interesante eliminar aquellos atributos que no sean relevantes. De esta forma existen técnicas que permiten obtener las variables más relevantes, aunque nuestro conocimiento sobre el dominio del problema puede permitirnos hacer correctamente muchas de esas selecciones.

Al igual que con las variables, también podríamos construir el modelo usando todos los datos. Pero si tenemos muchos, tardaríamos mucho tiempo y probablemente también necesitaríamos una máquina más potente. Consecuentemente, una buena idea es usar una muestra a partir de algunos datos (o filas). La selección de la muestra debe ser hecha cuidadosamente para asegurar que es verdaderamente aleatoria.

Otra tarea de preparación de los datos es la construcción de atributos, la cual consiste en construir automáticamente nuevos atributos aplicando alguna operación o función a los atributos originales con objeto de que estos nuevos atributos hagan más fácil el proceso de minería.

El tipo de los datos puede también modificarse para facilitar el uso de técnicas que requieren tipos de datos específicos. Así, algunos atributos se pueden numerizar, lo que reduce el espacio y permite usar técnicas numéricas. Y otros se pueden discretizar, es decir, transformar los valores numéricos en atributos discretos o nominales.

4.4.2.2. Fase de minería de datos

La fase de minería de datos es la más característica del KDD y, por esta razón, muchas veces se utiliza esta fase para nombrar todo el proceso. El objetivo de esta fase es producir nuevo conocimiento que pueda utilizar el usuario. Esto se realiza construyendo un modelo basado en los datos recopilados para este efecto. El modelo es una descripción de los patrones y relaciones entre los datos que pueden usarse para hacer predicciones, para entender mejor los datos o para explicar situaciones pasadas. Para ello es necesario tomar una serie de decisiones antes de empezar el proceso:

- Determinar qué tipo de tarea de minería es el más apropiado. Por ejemplo, podríamos usar la clasificación para predecir en una entidad bancaria los clientes que dejarán de serlo.
- Elegir el tipo de modelo. Por ejemplo, para una tarea de clasificación podríamos usar un árbol de decisión, porque queremos obtener un modelo en forma de reglas.
- Elegir el algoritmo de minería que resuelva la tarea y obtenga el tipo de modelo que estamos buscando. Esta elección es pertinente porque existen muchos métodos para construir los modelos.

En lo que resta de esta sección, describimos las tareas y modelos más utilizados, así como algunos conceptos relacionados con la construcción del modelo.

4.4.2.3. Fase de evaluación e interpretación

Medir la calidad de los patrones descubiertos por un algoritmo de minería de datos no es un problema trivial, ya que esta medida puede atañer a varios criterios, algunos de ellos bastante subjetivos. Idealmente, los patrones descubiertos deben tener tres cualidades: ser precisos, comprensibles (es decir, inteligibles) e interesantes (útiles y novedosos). Según las aplicaciones puede interesar mejorar algún criterio y sacrificar ligeramente otro, como en el caso del diagnóstico médico que prefiere patrones comprensibles aunque su precisión no sea muy buena.

Capítulo 5

KEPS: Un marco general para el diseño de metaheurísticas cooperativas

Se propone un sistema metaheurístico híbrido, paralelo, adaptativo y cooperativo basado en un sistema multiagente, al cual se le ha dado el nombre de KEPS. En este sistema diversas metaheurísticas, tanto basadas en poblaciones (algoritmos genéticos, optimización por enjambre de partículas, colonia de hormígas...) como basadas en trayectorias (búsqueda tabú, temple simulado, búsqueda por entornos variables...) cooperan para encontrar la mejor solución posible a una instancia de un problema de optimización. Durante su ejecución el sistema elije de manera inteligente los parámetros de cada una de las metaheurísticas, las cuales se ejecutan *paralelamente* mientras *cooperan* intercambiando sus soluciones de un modo *adaptativo*. La adaptatividad se consigue mediante el uso de un conjunto de reglas fuzzy capaz de evaluar la información extraída por un proceso de extracción de conocimiento preliminar.

La naturaleza paralela y distribuida de la aproximación es totalmente apropiada para ser modelada usando un sistema multiagente, donde un conjunto de *agentes de optimización* ejecutan las metaheurísticas cooperativas bajo la supervisión de un *agente coordinador*, cuya inteligencia es proporcionada por las reglas y el proceso de minería antes mencionados.

Tres aspectos importantes se deben tener en cuenta a la hora de definir la estrategia: 1) la arquitectura del sistema, 2) el esquema de cooperación que crea la sinergia entre las distintas metaheurísticas y 3) el modo en que se otorga adaptabilidad al sistema mediante el uso de un proceso de extracción del conocimiento.

5.1. Arquitectura

En esta sección se describirán dos arquitecturas distintas que se utilizarán más adelante. La primera, que denominaremos cooperativa bicapa, distingue dos fases de ejecución. Inicial-

mente distintas metaheurísticas basadas en poblaciones cooperan para explorar el espacio de búsqueda con el objetivo de localizar la zona más prometedora del mismo, a continuación, en la segunda fase, un conjunto de metaheurísticas basadas en trayectorias se encarga de explotar, en modo cooperativo, esta región y encontrar soluciones de alta calidad. La segunda arquitectura, llamada monocapa, propone un sistema en el que tanto metaheurísticas basadas en trayectorias, como basadas en poblaciones cooperan paralelamente. Con intención de diferenciar cuándo el sistema se compone de una o dos capas se utiliza un superíndice para indicar el número de capas, de tal manera que KEPS^2 hace referencia a la arquitectura bicapa y KEPS^1 a la monocapa.

5.1.1. Arquitectura bicapa

Sea P un problema de optimización y q una instancia dada de P. La arquitectura bicapa, KEPS^2, trata de resolver q usando una colección $M = E \cup L$ de metaheurísticas, donde $E = \{m_1, m_2, ..., m_\varepsilon\}$ y $L = \{m_{\varepsilon+1}, m_{\varepsilon+2}, ..., m_{\varepsilon+\iota}\}$ son metaheurísticas basadas en poblaciones y metaheurísticas basadas en trayectorias respectivamente. La ejecución de las metaheurísticas comprendidas en M se realiza en dos fases secuenciales que tratan respectivamente con E y L. Tanto las metaheurísticas basadas en poblaciones como las basadas en trayectorias son ejecutadas usando el paradigma de agentes, es decir, utilizando una colección de agentes $A = \{a_1, a_2, \ldots, a_M\}$ donde el subconjunto de agentes $\{a_1, a_2, \ldots, a_\varepsilon\}$ está relacionado con metaheurísticas basadas en poblaciones, mientras que el subconjunto $\{a_{\varepsilon+1}, a_{\varepsilon+2}, \ldots, a_M\}$ está relacionado con metaheurísticas basadas en trayectorias. El agente coordinador a^c es responsable de iniciar el proceso de optimización activando en modo paralelo los agentes de optimización, y en cada fase, coordina la cooperación entre los agentes de optimización de un modo adaptativo eligiendo valores para sus parámetros y decidiendo cuándo deben intercambiar soluciones.

Después de que el agente realice la etapa de inicialización, los agentes de optimización ejecutan en modo asincrónico sus metaheurísticas mientras envían y reciben información. El intercambio de información se realiza usando un esquema de pizarra. Concretamente se utilizan dos pizarras, la primera es usada por los agentes de optimización para publicar información acerca de su comportamiento, la segunda es usada por el coordinador para indicar sus instrucciones a cada agente de optimización. Específicamente, después de un periodo determinado, ya sea medido en tiempo o en evaluaciones de la función objetivo, cada metaheurística para y publica su solución/población (dependiendo de si se trata de una metaheurística basada en poblaciones o en trayectorias). Entonces el agente coordinador analiza el comportamiento de cada metaheurística, teniendo en cuenta conocimiento previo, y decide si alguna metaheurísti-

ca necesita mejorar su rendimiento. En ese caso, el coordinador escribe en la segunda pizarra la nueva solución/población que tendrá que usar la metaheurística. Los algoritmos 7 y 8 muestran, respectivamente, el comportamiento de los agentes coordinador y de optimización usando pseudo-código, mientras que la figura 5.1 muestra gráficamente la arquitectura.

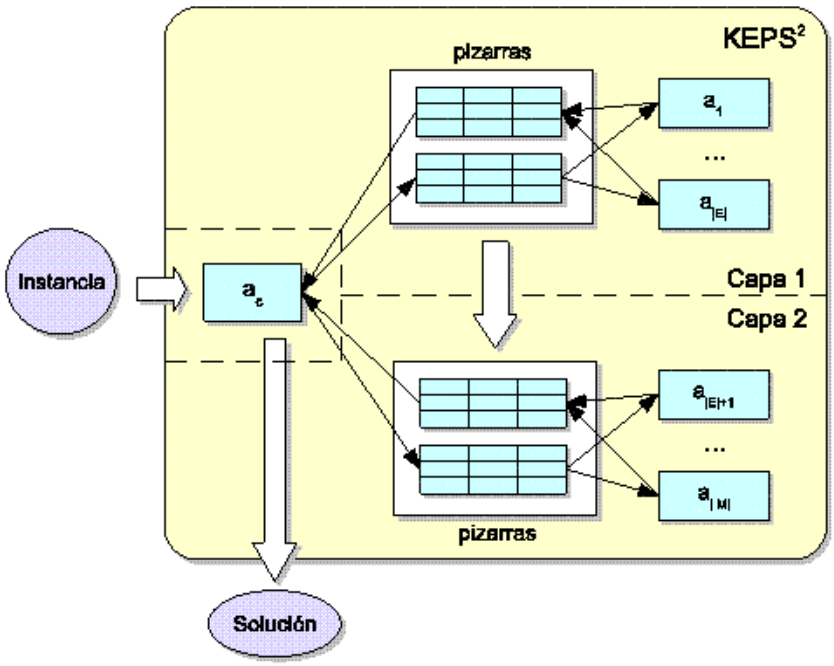


Figura 5.1: Arquitectura bicapa.

Algoritmo 7: Pseudo-código del agente de optimización.

Input: q: Instancia del problema de optimización,m_i: Metaheurística asignada al agente,n: periodo de ejecución

begin
  utilizar los valores de parámetros indicados por el coordinador;
  while *no se satisfaga la condición de fin* **do**
    leer *instrucciones* de la pizarra;
    if *se reciben instrucciones* **then**
      reemplazar la solución/población actual por aquella contenida en *instrucciones*;
    end
  end
  solucion=ejecutar metaheurística m_i durante n;
  escribir *solucion* en la pizarra;
end

Algoritmo 8: Pseudo-código del agente coordinador.

Input: n: periodo de ejecución para cada agente de optimización, q: instancia del problema, $M = E \cup L$: conjunto de metaheurísticas
Output: *solucion*: mejor solución encontrada
begin
  seleccionar los valores de parámetros para cada metaheurística en E;
  while *no se satisfaga la condición de fin* **do**
    ejecutar en paralelo cada agente de $\{a_1, \ldots, a_E\}$;
    utilizar el conocimiento disponible, y los datos de la pizarra para seleccionar $POOR$, un conjunto de agentes que obtienen bajo rendimiento;
    foreach *agente a in POOR* **do**
      crear $instrucciones_a$ que contengan una población más apropiada;
      escribir $instrucciones_a$ en la pizarra;
    end
  end
  seleccionar los valores de parámetros para cada metaheurística en L;
  mejor_solucion=mejor solución encontrada por los agentes de E;
  utilizar *mejor_solucion* como solución inicial para los agentes de L;
  while *no se satisfaga la condición de fin* **do**
    ejecutar en paralelo cada agente de $\{a_{E+1}, \ldots, a_M\}$;
    utilizar el conocimiento disponible y los datos de la pizarra para seleccionar $POOR$, un conjunto de agentes que obtienen bajo rendimiento;
    foreach *agente a in POOR* **do**
      crear $instrucciones_a$ que contengan una solución más apropiada;
      escribir $instrucciones_a$ en la pizarra;
    end
  end
  devolver la mejor solución encontrada por los agentes de optimización.;
end

5.1.2. Arquitectura monocapa

La arquitectura monocapa, KEPS^1, es similar a la anterior, salvo por el hecho de que los agentes $a_i \in A$ independientemente de si ejecutan una metaheurística perteneciente a E o a L son ejecutados en paralelo y cooperan todos entre sí.

A diferencia de la arquitectura bicapa, en la arquitectura monocapa la interfaz entre metaheurísticas basadas en poblaciones y basadas en trayectorias no es simplemente el paso de la mejor solución, sino que se deben definir políticas de intercambio de soluciones en ambas direcciones, que se explicarán con mayor detenimiento en la siguiente sección.

La nueva arquitectura se muestra en la figura 5.2 de igual manera, el algoritmo de a^c se muestra en el algoritmo 9, puesto que cambia ligeramente, sin embargo el algoritmo de los a_i se mantiene invariable.

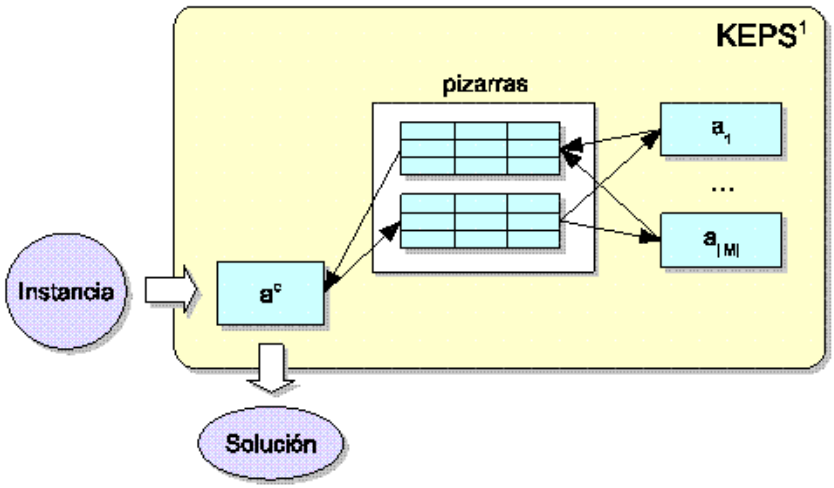


Figura 5.2: Arquitectura de una capa.

Después de introducir las dos arquitecturas propuestas se afronta otra cuestión crucial en el diseño de la estrategia cooperativa, el esquema de cooperación, que se presentará en la siguiente sección.

Algoritmo 9: Pseudo-código del agente coordinador.

Input: n: periodo de ejecución para cada agente de optimización, q: instancia del problema, $M = E \cup L$: conjunto de metaheurísticas
Output: *solucion*: mejor solución encontrada
begin
  seleccionar los valores de parámetros para cada metaheurística en M;
  while *no se satisfaga la condición de fin* **do**
    ejecutar en paralelo cada agente de $\{a_1, \ldots, a_M\}$;
    utilizar las reglas fuzzy y los datos de la pizarra para seleccionar $POOR$, un conjunto de agentes que obtienen bajo rendimiento;
    foreach *agente a in* $POOR$ **do**
      crear $instrucciones_a$ que contengan una población más apropiada;
      escribir $instrucciones_a$ en la pizarra;
    end
  end
  devolver la mejor solución encontrada por los agentes de optimización.;
end

5.2. Esquema de cooperación

El esquema de cooperación de la estrategia está definido principalmente por las reglas de control usadas por el coordinador y en su caso los modelos de conocimiento que guían su mecanismo de adaptación. Estos componentes deben ayudar a a^c a:

- Elegir los mejores valores para los parámetros de cada metaheurística implementada por un agente $a_i \in M$.

- Descubrir cuándo una metaheurística con un comportamiento pobre tiene que recibir nuevas soluciones de otras metaheurísticas que muestran mejor comportamiento.

Se estudian dos esquemas posibles. El primero, propuesto en [214], se basa en el uso de la memoria obtenida durante la ejecución. El segundo, que es el que se propone, se basa en el conocimiento extraído de ejecuciones previas de cada metaheurística. Con la intención de distinguir los distintos esquemas de cooperación utilizados por la metaheurística se propone indicar como subíndice el esquema de cooperación, de tal manera que $\text{KEPS}_{\{\bullet\}}$ representa que el esquema utilizado por KEPS está basado en el esquema $\{\bullet\}$. En las siguientes subsecciones pasamos a explicar cada uno de estos esquemas de cooperación.

5.2.1. Esquema basado en memoria

El esquema basado en memoria, que denominamos Mem (la metaheurística KEPS con el esquema Mem la denotamos por KEPS_{Mem}), fue diseñado siguiendo el principio: "Si un agente de optimización está trabajando bien, mantenlo; pero si un agente de optimización parece atrapado, haz algo para alterar su comportamiento".

Para la primera tarea del coordinador, la elección de los valores de los parámetros, utiliza un enfoque "ad-hoc", es decir, un experto debe escogerlos y permanecerán invariables independientemente de la instancia que se esté resolviendo.

Por otro lado, la segunda tarea, utiliza dos estructuras de memoria de tamaño fijo y una regla fuzzy para escoger durante la ejecución qué metaheurísticas deben modificar su comportamiento.

La primera estructura de memoria, llamada memoria de calidades, almacena ordenadamente medidas de la calidad de las soluciones obtenidas hasta el momento. Mientras que la segunda, llamada memoria de ratios, almacena, también ordenadamente, medidas del ratio de mejora obtenido por las metaheurísticas, estando calculado el ratio de mejora como:

$$\Delta f = \frac{f(s^t) \quad f(s^{t'})}{t \quad t'}$$

donde:

- $f(x)$ es una función que valora la calidad de una solución.
- s^t es la solución obtenida en el momento t.
- t es una medida del tiempo en el momento en que se envía la solución.
- t' es una medida del tiempo en el momento en que se envió la última solución anterior a s^t.

La regla fuzzy es la siguiente:

si $f(s_i^t)$ es $baja$ y Δf_i es $bajo$ **entonces** enviar solución cercana a C_{best} a a_i

donde:

- $f(s_i)$ es la calidad de la solución enviada por a_i.
- Δf_i es el índice de mejora conseguido por el agente a_i.
- $baja$ y $bajo$ son conjuntos fuzzy cuya función de pertenencia, $\mu(x)$, tiene una forma trapezoidal definida por la cuádrupla $(\ -\infty,\ -\infty, 80, 100)$. La variable x corresponde a la posición relativa (similar a la noción de rango percentil) de un valor en la memoria de calidades.

- C_{best} denota la mejor solución encontrada por el coordinador.

La regla indica que si los valores reportados por a_i están entre los peores almacenados en las dos memorias, entonces el coordinador debe enviarle la solucion C_{best} ligeramente modificada. Haciendo esto, reposiciona a a_i en una región más prometedora del espacio de búsqueda, intentando mejorar las probabilidades de encontrar mejores soluciones. La modificación de C_{best} se suele realizar con un operador de mutación.

Para realizar el disparo de las reglas fuzzy, se utilizan α-cortes. En otras palabras, solo aquellas reglas cuyo valor de activación exceda el α-corte definido serán disparadas (se puede ver un ejemplo del disparo de las reglas en la fig. 5.3). En el caso de que más de una regla

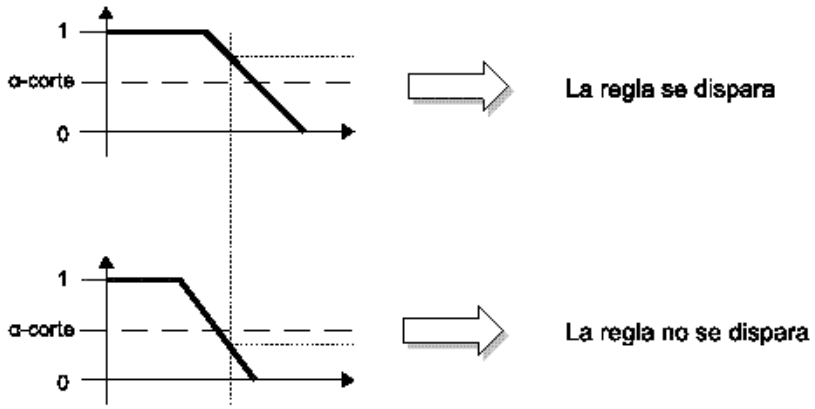


Figura 5.3: Ejemplo α-cortes.

sea activada, a^c las aplica todas. De esta manera es capaz de adaptar el comportamiento de varias metaheurísticas con bajo rendimiento de manera simultánea.

Ejemplo 5.1 (Proceso de disparo de las reglas fuzzy basadas en memoria)**.** En este ejemplo se pretende ilustrar el proceso de disparo de las reglas fuzzy basadas en memoria. Supongamos que las memorias de calidades y ratios son las mostradas en la tabla 5.1, por lo que el conjunto fuzzy *baja* tomará los valores $(-\infty, -\infty, 1792, 1912)$ y el conjunto *bajo* los valores $(-\infty, -\infty, 1.53, 1.76)$. Supongamos también que el α corte es 0.5 y los valores de calidad y solución mostrados en la tabla 5.2 para dos metaheurísticas, m_1 y m_2. Como puede observarse, la solución de m_1 tiene una calidad que activa parcialmente la clausula de calidad, exactamente con un 0.83, además su ratio de mejora activa totalmente la clausula relativa al ratio y por tanto la activación final será del 0.83, lo cual es mayor que 0.5 y por tanto la regla se dispara. Por el contrario en el caso de m_2 la calidad de la solución encontrada es superior a la mejor almacenada, y por tanto no se disparará la regla.

Tabla 5.1: Ejemplos de memoria de calidades y ratios

memoria de calidades	1231	1413	1683	1792	1912
memoria de ratios	0.86	0.92	1.28	1.53	1.76

Tabla 5.2: Tres posibles configuraciones diferentes

	m_1	m_2
calidad	1813	1925
ratio	1.11	0.76
activación de la regla	0.83	0.0
cambio de solución	si	no

■

5.2.2. Esquema basado en conocimiento previo

En este esquema, la adaptabilidad del coordinador se realiza teniendo en cuenta el conocimiento obtenido a través de un proceso de extracción inteligente del conocimiento, que utiliza modelos de minería de datos (como por ejemplo, árboles de decisión, máquinas de vectores de soporte, redes neuronales...). Denotamos por CP $\{\square\}$ al esquema basado en conomiento previo siendo $\{\square\}$ el tipo de modelo de representación utilizado para registrar el conocimiento adquirido. La mateheurística KEPS con este tipo de esquema la denotamos por $\text{KEPS}_{CP\ \{\square\}}$.

El análisis de los modelos permite a a^c elegir los mejores parámetros para cada metaheurística en función de la instancia q y ordenar la idoneidad de cada metaheurística para resolver q a través de pesos numéricos en el rango $[0, 1]$. En detalle, mediante el análisis de los pesos y del comportamiento que está mostrando cada metaheurística durante la ejecución, a^c es capaz de evaluar qué metaheurística está obteniendo resultados pobres y, consecuentemente, puede actualizar su solución/población enviándole nuevas soluciones provenientes de otras metaheurísticas que estén funcionando mejor. Este proceso permite a las metaheurísticas recomenzar su búsqueda en un punto más interesante del espacio de búsqueda.

La primera tarea del coordinador, elegir los valores de los parámetros de los agentes, es llevada a cabo a través de un conjunto de modelos de conocimiento obtenidos tras la aplicación de un proceso de aprendizaje supervisado. Cada uno de estos modelos está relacionado con un parámetro de una metaheurística, de tal modo que su uso para analizar una instancia dada q_{a_1} indica los mejores valores para los parámetros de cada una de las metaheurísticas.

La segunda tarea es realizada por a^c teniendo en cuenta otro conjunto de modelos de conocimiento. En detalle, estos modelos analizan una instancia dada q y ordenan las

diferentes metaheurísticas que participan en la cooperación, devolviendo una colección de pesos $\Omega = \{\omega_i | i = 1, \ldots, |M|\}$, donde $\omega_i \in [0, 1]$ y, en el caso de ser la arquitectura bicapa $\sum_{i=1}^{\mathcal{E}} \omega_i = 1$ y $\sum_{i=\mathcal{E}+1}^{M} \omega_i = 1$, mientras que para la arquitectura monocapa, $\sum_{i=1}^{M} \omega_i = 1$.

En resumen, cada peso ω_i está asociado con una metaheurística $m_i \in M$ y representa su idoneidad para resolver q. Más precisamente, sean $m_i, m_j \in |M|$ dos metaheurísticas, entonces $\omega_i > \omega_j$ implica que, de acuerdo con ejecuciones previas, la i-ésima metaheurística obtiene un rendimiento global mejor que el obtenido por la j-ésima metaheurística.

Para controlar el intercambio de soluciones a^c utiliza una colección de reglas fuzzy (una por cada metaheurística) que utilizan el conjunto de pesos Ω para derivar el conjunto de metaheurísticas que se están comportando pobremente:

si $(\omega_1 \cdot \delta_1)$ is *suficiente* **entonces** *enviar solución de* m_1 *a* m_h
si $(\omega_2 \cdot \delta_2)$ is *suficiente* **entonces** *enviar solución de* m_2 *a* m_h
...
si $(\omega_{h-1} \cdot \delta_{h-1})$ is *suficiente* **entonces** *enviar solución de* m_{h-1} *a* m_h
si $(\omega_{h+1} \cdot \delta_{h+1})$ is *suficiente* **entonces** *enviar solución de* m_{h+1} *a* m_h
...
si $(\omega_M \cdot \delta_M)$ is *suficiente* **entonces** *enviar solución de* m_M *a* m_h

donde:

- m_h es la metaheurística que se está evaluando;
- $\delta_i = (\xi(m_i) - \xi(m_h))/\max|\xi(m_i)|, |\xi(m_h)|$, donde ξ es una medida del rendimiento mostrado por las metaheurísticas, que es definida por el usuario;
- $\omega_i \in [0, 1]$ es el peso de m_i;
- $suficiente$ es un conjunto fuzzy con función de pertenencia trapezoidal definida por la cuadrupla (a, b, c, d) y cuyo universo de discurso es el rango $[0, 1]$.

El principal objetivo de cada regla fuzzy es cambiar la posición en el espacio de búsqueda de una metaheurística que esté mostrando un rendimiento pobre, por una posición cercana a la solución de otra metaheurística con un comportamiento mejor. Se debe notar que esta regla trata de resolver el problema que aparece en las estrategias paralelas, donde el intercambio de soluciones no está restringido [88] y que tienden a converger prematuramente a óptimos locales de baja calidad.

Para realizar el disparo de las reglas fuzzy, se utilizan α-cortes, como se indicó anteriormente. En el caso de que más de una regla sea activada, a^c las aplica todas. De esta manera es capaz de adaptar el comportamiento de varias metaheurísticas con bajo rendimiento de manera simultánea.

El intercambio de soluciones entre metaheurísticas es realizado por a^c teniendo en cuenta diversas situaciones. En detalle, sea $S \subset M$ la colección de metaheurísticas relacionadas con las reglas fuzzy que se han disparado, y sea m_h la metaheurística que presenta un comportamiento pobre y que debe recibir mejores soluciones. Si sólo se ha disparado una regla entonces $|S| = 1$, en otro caso $|S| > 1$. El intercambio de soluciones se realiza de la siguiente forma:

- Si m_h es basada en poblaciones, se pueden encontrar los siguientes casos:
  - $|S| = 1$ y $s \in S$ es basada en trayectorias. En este caso una proporción de los peores individuos de m_h igual a ω_s se sustituye por un conjunto de soluciones consistente en soluciones cercanas a la mejor solución obtenida por s, donde cercana significa que es cambiada aplicando $\lceil \frac{1}{(2 \cdot \omega_s)} \rceil$ veces un operador de mutación dependiente del problema P.
  - $|S| = 1$ y $s \in S$ es basada en poblaciones. En este caso una proporción de los individuos de m_h igual a ω_s se sustituye por un conjunto de soluciones consistente en los mejores individuos de la población de s.
  - $|S| > 1$ y $s_i \in S$, con $i = 1, \ldots, |S|$. En este caso una proporción de los individuos de m_h igual a $\sum_{i=1}^{S} \omega_i$ es sustituida por un conjunto de soluciones donde cada s_i selecciona qué soluciones enviará siguiendo los métodos antes descritos.
- Si m_h es basada en trayectorias:
  - Una solución cercana a la mejor solución obtenida por todas la metaheurísticas pertenecientes a S se envía a m_h, la cual sustituye a su solución actual.

Ejemplo 5.2 (Proceso de disparo de las reglas fuzzy basadas en conocimiento)**.** En este ejemplo se pretende ilustrar el proceso de disparo de las reglas fuzzy basadas en conocimiento. En él suponemos un problema de minimización, y dos metaheurísticas, m_1 y m_2, ($|M| = 2$), el conjunto fuzzy *suficiente* toma los valores $[0, 0.1, 1, 1]$, el valor del α-corte es 0.5 y $\xi =$ valor de la función objetivo. La tabla 5.3 muestra tres posibles configuraciones diferentes.

Tabla 5.3: Tres posibles configuraciones diferentes

	m_1	m_2	m_1	m_2	m_1	m_2
peso	0.43	0.57	0.68	0.32	0.55	0.45
valor de la función objetivo	178	154	178	154	178	154
activación de la regla	0.77	0	0.43	0	0.61	0
cambio de solución	si	no	no	no	si	no

En el primer caso, de acuerdo con los pesos ω_i obtenidos del modelo de conocimiento, m_2 tiene un comportamiento global mejor que el de m_1 para la instancia estudiada, además

está obteniendo una mejor solución en el momento. Recordemos que la entrada de la regla es $\omega_i \cdot \delta_i$ donde $\delta_i = (\xi(m_i) \quad \xi(m_h)/\max\xi(m_i), \xi(m_h)$. Por tanto cuando a^c evalúa la regla para m_1, obtiene un $\delta_i = (\ 178 + 154)/178 = 1,35$ y el valor de entrada a la regla será $1,35 \times 0,57 = 0,77$ lo que activa la regla con un valor de activación del 0.77 ($> \alpha$-corte), lo que significa que m_1 debe recibir soluciones de m_2 para mejorar su comportamiento. Por otro lado, la regla para m_2 no se dispara. En el segundo ejemplo, m_1 presenta un mejor comportamiento global que m_2 (es decir $\omega_{m_1} > \omega_{m_2}$), pero las soluciones se mantienen. En este caso, cuando la regla se evalúa para m_1, a pesar de que su solución actual es peor que la de m_2, no se dispara. De hecho, de acuerdo con los pesos, m_1 es mucho mejor que m_2 para esta instancia específica. De este modo, se evita una convergencia prematura a soluciones locales de baja calidad. El último ejemplo es similar al anterior, pero la diferencia de pesos no es tan marcada, y por tanto la regla se dispara para m_1, puesto que se tiene más confianza en el rendimiento de m_2.

■

Para obtener los modelos de parámetros y pesos se utiliza un proceso de extracción inteligente del conocimiento como el que se muestra en la fig. 5.4, compuesto de dos fases diferentes, preparación de datos y minería de datos, que serán explicadas con más detenimiento en la siguiente sección. Se debe notar que este proceso es un proceso de aprendizaje supervisado y que sólo es necesario aplicarlo una vez, tras lo cual el sistema puede ser usado para resolver cualquier instancia del problema considerado.

5.3. Otorgando adaptabilidad al sistema

En esta sección se explica detalladamente el proceso de extracción del conocimiento que dará adaptabilidad a $\text{KEPS}_{C^P\ \{\square\}}$.

Sin pérdida de generalidad, sea P un problema de maximización, I un conjunto de instancias de P, utilizadas como instancias de entrenamiento y caracterizadas por distintos atributos del problema $F_P = \{f_1, f_2, \ldots, f_o\}$, y $M = E \cup L$ una colección de $|E|$ metaheurísticas basadas en poblaciones y $|L|$ metaheurísticas basadas en trayectorias, cuya ejecución depende de una colección de n_i parámetros $P_i = \{p_i^1, p_i^2, \ldots, p_i^{n_i}\}$. El objetivo del proceso de extracción del conocimiento es evaluar los rendimientos de las metaheurísticas en M cuando son aplicadas a instancias en I. Este proceso devuelve un conjunto de modelos $T = \{t_{p_1^1}, \ldots, t_{p_1^{n_1}}, \ldots, t_{p_{|M|}^1}, \ldots, t_{p_{|M|}^{n_{|M|}}}\} \cup \{t^{\omega_1}, \ldots, t^{\omega_{|M|}}\}$ que contiene $\sum_{i=1}^{M} |P_i| + |M|$ modelos de minería de datos, donde t^{ω_i} modela la idoneidad de una metaheurística $m_i \in M$ para la resolución del problema P. Mientras que cada modelo $t_{p_i^j}$, con $i = 1, \ldots |M|$ y $j = 1, \ldots, n_i$,

proporciona información acerca de la elección idónea del parámetro p_i^j para la metaheurística m_i aplicada a la solución de P.

El proceso de extracción del conocimiento se subdivide en dos subfases, como se puede ver en la fig. 5.4, preparación de los datos y minería de datos.

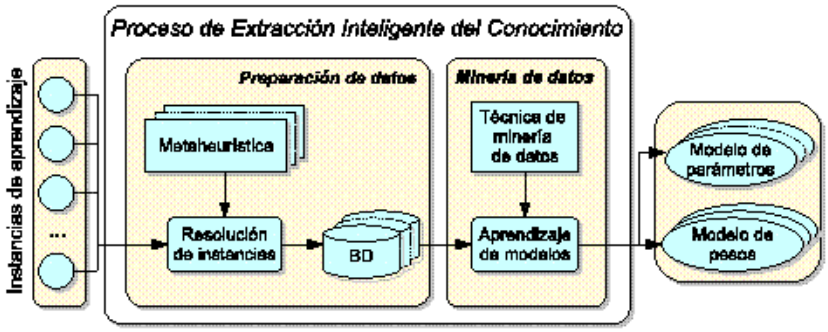


Figura 5.4: Proceso de extracción inteligente del conocimiento.

5.3.1. Preparación de los datos

En esta fase, cada metaheurística $m_i \in M$ se aplica a la cada instancia $q \in I$ con el objetivo de construir una colección de bases de datos que se utilizarán en la fase de minería para obtener T. En particular, sea m_i una metaheurística genérica en M cuya ejecución depende de una colección de n_i parámetros $P_i = \{p_i^1, p_i^2, \ldots, p_i^{n_i}\}$ y sea $V_{i,j} = \{v_{i,j}^1, v_{i,j}^2, \ldots, v_{i,j}^{s_{i,j}}\}$, con $j = 1, \ldots, n_i$, una colección finita de $s_{i,j}$ valores posibles para p_i^j, el j ésimo parámetro de la i ésima metaheurística. Consecuentemente, si $c_i = \prod_{l=1}^{n_i} |V_{i,l}|$ es el número de combinaciones de valores de parámetros (es decir, la cardinalidad del conjunto producto $V_{i,1} \times V_{i,2} \times \ldots \times V_{i,n_i}$) entonces la fase de preparación de los datos resuelve cada instancia $q \in I$ aplicando $r = c_i \cdot k$ veces la metaheurística m_i, donde k es un *factor iterativo*, es decir, un valor predeterminado usado para obtener estimaciones de rendimiento más precisas. El proceso de extracción del conocimiento utiliza estimaciones de rendimiento para rellenar $r \cdot |I|$ filas de la base de datos relativa a la metaheurística m_i. En particular, una fila de la base de datos contiene:

- Una descripción de la instancia específica del problema.
- Los valores de los parámetros usados en la ejecución de la metaheurística.
- La solución final obtenida.

La tabla 5.4 muestra las entradas de la base de datos relacionadas con la metaheurística m_i. Las bases de datos obtenidas usando este proceso son denominadas *bases de datos crudas*.

Tabla 5.4: BD relacionada con la metaheurística m_i. Cada instancia q_i caracterizada por los atributos $\{f_1, f_2, \ldots, f_o\}$, m_i es resuelta k veces con cada combinación de valores de parámetros. $fobj_z^u$ representa el z-ésimo valor de la función objetivo.

			Attributes				Parameters				Final
			f_1	f_2	...	f_o	p_i^1	p_i^2	...	$p_i^{n_i}$	Solution
Instance	Par. Comb.	It.									
q_1	1	1	f_1^1	f_2^1	...	f_o^1	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^1	$fobj_1^1$
		2	f_1^1	f_2^1	...	f_o^1	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^1	$fobj_2^1$
		...	...	...	...	...	...	...	...	...	...
		k	f_1^1	f_2^1	...	f_o^1	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^1	$fobj_k^1$
	2	1	f_1^1	f_2^1	...	f_o^1	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^2	$fobj_{k+1}^1$
		2	f_1^1	f_2^1	...	f_o^1	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^2	$fobj_{k+2}^1$
		...	...	...	...	...	...	...	...	...	...
		k	f_1^1	f_2^1	...	f_o^1	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^2	$fobj_{k+k}^1$
	...	...	...	...	...	...	...	...	...	...	...
	...	...	...	...	...	...	...	...	...	...	...
	c_i	1	f_1^1	f_2^1	...	f_o^1	$v_{i,1}^{x_{i,1}}$	$v_{i,2}^{x_{i,2}}$	...	$v_{i,n_i}^{x_{i,n_i}}$	$fobj_{(c_i-1)\cdot k+1}^1$
		2	f_1^1	f_2^1	...	f_o^1	$v_{i,1}^{x_{i,1}}$	$v_{i,2}^{x_{i,2}}$	...	$v_{i,n_i}^{x_{i,n_i}}$	$fobj_{(c_i-1)\cdot k+2}^1$
		...	...	...	...	...	...	...	...	...	...
		k	f_1^1	f_2^1	...	f_o^1	$v_{i,1}^{x_{i,1}}$	$v_{i,2}^{x_{i,2}}$	...	$v_{i,n_i}^{x_{i,n_i}}$	$fobj_{c_i\cdot k}^1$
q_2	1	1	f_1^2	f_2^2	...	f_o^2	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^1	$fobj_1^2$
		2	f_1^2	f_2^2	...	f_o^2	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^1	$fobj_2^2$
		...	...	...	...	...	...	...	...	...	...
		k	f_1^2	f_2^2	...	f_o^2	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^1	$fobj_k^2$
	2	1	f_1^2	f_2^2	...	f_o^2	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^2	$fobj_{k+1}^2$
		2	f_1^2	f_2^2	...	f_o^2	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^2	$fobj_{k+2}^2$
		...	...	...	...	...	...	...	...	...	...
		k	f_1^2	f_2^2	...	f_o^2	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^2	$fobj_{k+k}^2$
	...	...	...	...	...	...	...	...	...	...	...
	...	...	...	...	...	...	...	...	...	...	...
	c_i	1	f_1^2	f_2^2	...	f_o^2	$v_{i,1}^{x_{i,1}}$	$v_{i,2}^{x_{i,2}}$	...	$v_{i,n_i}^{x_{i,n_i}}$	$fobj_{(c_i-1)\cdot k+1}^2$
		2	f_1^2	f_2^2	...	f_o^2	$v_{i,1}^{x_{i,1}}$	$v_{i,2}^{x_{i,2}}$	...	$v_{i,n_i}^{x_{i,n_i}}$	$fobj_{(c_i-1)\cdot k+2}^2$
		...	...	...	...	...	...	...	...	...	...
		k	f_1^2	f_2^2	...	f_o^2	$v_{i,1}^{x_{i,1}}$	$v_{i,2}^{x_{i,2}}$	...	$v_{i,n_i}^{x_{i,n_i}}$	$fobj_{c_i\cdot k}^2$
	...	...	...	...	...	...	...	...	...	...	...
	...	...	...	...	...	...	...	...	...	...	...
	...	...	...	...	...	...	...	...	...	...	...
	...	...	...	...	...	...	...	...	...	...	...
q_l	1	1	f_1^l	f_2^l	...	f_o^l	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^1	$fobj_1^l$
		2	f_1^l	f_2^l	...	f_o^l	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^1	$fobj_2^l$
		...	...	...	...	...	...	...	...	...	...
		k	f_1^l	f_2^l	...	f_o^l	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^1	$fobj_k^l$
	2	1	f_1^l	f_2^l	...	f_o^l	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^2	$fobj_{k+1}^l$
		2	f_1^l	f_2^l	...	f_o^l	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^2	$fobj_{k+2}^l$
		...	...	...	...	...	...	...	...	...	...
		k	f_1^l	f_2^l	...	f_o^l	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^2	$fobj_{k+k}^l$
	...	...	...	...	...	...	...	...	...	...	...
	...	...	...	...	...	...	...	...	...	...	...
	c_i	1	f_1^l	f_2^l	...	f_o^l	$v_{i,1}^{x_{i,1}}$	$v_{i,2}^{x_{i,2}}$	...	$v_{i,n_i}^{x_{i,n_i}}$	$fobj_{(c_i-1)\cdot k+1}^l$
		2	f_1^l	f_2^l	...	f_o^l	$v_{i,1}^{x_{i,1}}$	$v_{i,2}^{x_{i,2}}$	...	$v_{i,n_i}^{x_{i,n_i}}$	$fobj_{(c_i-1)\cdot k+2}^l$
		...	...	...	...	...	...	...	...	...	...
		k	f_1^l	f_2^l	...	f_o^l	$v_{i,1}^{x_{i,1}}$	$v_{i,2}^{x_{i,2}}$	...	$v_{i,n_i}^{x_{i,n_i}}$	$fobj_{c_i\cdot k}^l$
			Raw Database Entries								

Ejemplo 5.3. Sea m_i un algoritmo genético, entonces los conjuntos de parámetros se pueden definir como:

$$\begin{aligned} P_i &= \{p_i^1, p_i^2, p_i^3 | p_i^1, p_i^2 \in [0,1] \text{ and } p_i^3 \in \mathbb{N}\} \\ V_i^1 &= \{0.1, 0.3, 0.7\} \\ V_i^2 &= \{0.01, 0.1, 0.3\} \\ V_i^3 &= \{10, 40\} \end{aligned} \tag{5.1}$$

donde p_i^1 es la probabilidad de cruce, p_i^2 es la probabilidad de mutación y p_i^3 es el número de individuos. Entonces, el número de combinaciones de valores de parámetros es $|V_{i,1}| \times |V_{i,2}| \times |V_{i,3}| = 18$. Ahora, si $k = 5$ entonces el proceso de preparación de los datos resuelve cada instancia $q \in I$ aplicando, $k \cdot 18 = 54$ veces, el algoritmo genético. Los datos recolectados llenan $54 \cdot |I|$ filas en la base de datos del algoritmo genético.

□

El siguiente paso es analizar y refinar las bases de datos crudas para obtener una nueva colección de bases de datos $DB = \{d_{m_1}, d_{m_2}, \ldots, d_{m_{|M|}}, d_w\}$, con $|DB| = |M| + 1$, que contiene las llamadas *bases de datos refinadas*. Estas bases de datos son más pequeñas que las crudas y contienen información adicional útil para extraer conocimiento significativo durante la fase de minería de datos. En detalle, cada d_{m_i}, relacionada con la metaheurística $m_i \in M$, contiene información acerca de las combinaciones de valores de parámetros más apropiadas para resolver las instancias en I. Por otro lado, d_w contiene información acerca de la idoneidad de cada metaheurística en M.

Para construir las bases de datos en $\{d_{m_i} | i = 1, \ldots, |M|\}$, para cada metaheurística $m_i \in M$ se procesa su base de datos del siguiente modo:

1. Para cada instancia $q_a \in I$ y para cada combinación de parametros $pc_b^i \in V_{i,1} \times V_{i,2} \times \ldots \times V_{i,n_i}$, $b = 1, \ldots, c_i$, se calcula el valor de la función objetivo medio obtenido $med_b^{a,i} = \frac{\sum_{u=1+(b-1)\cdot k}^{b \cdot k} f obj_u^a)}{k}$.

2. Para cada instancia $q_a \in I$, se define $pc_{best}^{a,i} \in \{pc_b^i | b = 1, \ldots, c_i\}$ como la combinación de parámetros que obtiene el mejor valor de función objetivo medio, $max_{b=1}^{c_i} med_b^{a,i}$, y actualizar la base de datos refinada d_{m_i} con una nueva entrada que contiene la descripción de la instancia q_a y la mejor combinación de valores de parámetros, es decir, $(f_1^a, f_2^a, \ldots, f_o^a)$ y $pc_{best}^{a,i}$.

La otra base de datos, d_w, se encarga de proporcionar información acerca de la idoneidad de cada metaheurística. Para su construcción se deben seguir los siguientes pasos:

- Para cada metaheurística $m_i \in M$, para cada instancia $q_a \in I$ y para cada combinación de parámetros $pc_b^i \in V_{i,1} \times V_{i,2} \times \ldots \times V_{i,n_i}$, se calcula el valor medio obtenido de la función objetivo $med_b^{a,i} = max_{u=1+(b-1)\cdot k}^{b\cdot k}(fobj_u^a)$.

- Para cada $m_i \in M$ y $q_a \in I$, se selecciona el mejor valor medio de función objetivo $med_{best}^{a,i} = max_{u=1+(b-1)\cdot k}^{b\cdot k}(fo_u^a)$ y se calcula el *peso de idoneidad de la metaheurística i* como:

$$peso_i = \frac{fobj_{best}^{a,i}}{\sum_{l=1}^{E} fobj_{best}^{a,l}}$$

- Para cada $m_i \in E$ y para cada instancia $q_a \in I$ la base de datos refinada se actualiza con una nueva entrada que contiene una descripción de q_a y su peso de idoneidad, es decir, $(f_1^a, f_2^a, \ldots, f_o^a)$ y $peso_i$.

La tabla 5.5 muestra el proceso de refinamiento.

Tabla 5.5: Obtención de las bases de datos refinada a través de las crudas.

			Attributes				Parameters				Final
			f_1	f_2	...	f_o	p_i^1	p_i^2	...	$p_i^{n_i}$	Solution
Instance	Par. Comb.	It.									
...	...	...	...	...	...	...	...	...	...	...	...
q_a	1	1	f_1^a	f_2^a	...	f_o^a	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^1	$fobj_1^a$
		2	f_1^a	f_2^a	...	f_o^a	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^1	$fobj_2^a$
		...	...	...	...	...	...	...	...	...	...
		k	f_1^a	f_2^a	...	f_o^a	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^1	$fobj_k^a$
	2	1	f_1^a	f_2^a	...	f_o^a	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^2	$fobj_{k+1}^a$
		2	f_1^a	f_2^a	...	f_o^a	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^2	$fobj_{k+2}^a$
		...	...	...	...	...	...	...	...	...	...
		k	f_1^a	f_2^a	...	f_o^a	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^2	$fobj_{k+k}^a$
	...	...	...	...	...	...	...	...	...	...	...
	...	...	...	...	...	...	...	...	...	...	...
	c_i	1	f_1^a	f_2^a	...	f_o^a	$v_{i,1}^{x_{i,1}}$	$v_{i,2}^{x_{i,2}}$	...	$v_{i,n_i}^{x_{i,n_i}}$	$fobj_{(c_i-1)\cdot k+1}^a$
		2	f_1^a	f_2^a	...	f_o^a	$v_{i,1}^{x_{i,1}}$	$v_{i,2}^{x_{i,2}}$	...	$v_{i,n_i}^{x_{i,n_i}}$	$fobj_{(c_i-1)\cdot k+2}^a$
		...	...	...	...	...	...	...	...	...	...
		k	f_1^a	f_2^a	...	f_o^a	$v_{i,1}^{x_{i,1}}$	$v_{i,2}^{x_{i,2}}$	...	$v_{i,n_i}^{x_{i,n_i}}$	$fobj_{c_i\cdot k}^a$
...	...	...	...	...	...	...	...	...	...	...	...
			↓ ↓ fobjness Average Computation ↓ ↓								
...	...	...	...	...	...	...	...	...	...	...	...
	1	avg.	f_1^a	f_2^a	...	f_o^a	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^1	$avg_1^{a,i}$
	2	avg.	f_1^a	f_2^a	...	f_o^a	$v_{i,1}^1$	$v_{i,2}^1$	...	v_{i,n_i}^2	$avg_2^{a,i}$
q_a	...	...	...	...	...	...	...	...	...	...	...
	...	...	...	...	...	...	...	...	...	...	...
	c_i	avg.	f_1^a	f_2^a	...	f_o^a	$v_{i,1}^{x_{i,1}}$	$v_{i,2}^{x_{i,2}}$	...	$v_{i,n_i}^{x_{i,n_i}}$	$avg_{c_i}^{a,i}$
...	...	...	...	...	...	...	...	...	...	...	...

↓ ↓ Database Entry Creation ↓ ↓

f_1^a	f_2^a	...	f_o^a	$pc_{best}^{a,i}$	f_1^a	f_2^a	...	f_o^a	$suit_i$
Entry of Database d_{m_i}					Entry of Database d_L or d_E				

Las bases de datos refinadas representan el resultado final de la fase de preparación de los datos y serán usadas en la siguiente fase para completar el proceso de extracción inteligente del conocimiento.

En resumen, la preparación de los datos se aplica a las metaheurísticas de M para crear una colección de $|M|$ bases de datos crudas, que contienen $|I| \cdot c_i \cdot k$ filas. El proceso de refinamiento construye $|M| + 1$ bases de datos refinadas, de las cuales, cada base de datos contenida en el subconjunto $\{d_{m_i} | i = 1, \ldots, |M|\}$ contiene $|I|$ entradas, mientras que la base de datos d_ω contiene $|I| \cdot |M|$ entradas. Consecuentemente, si se utilizaran las bases de datos crudas, una técnica de minería de datos genérica analizaría $\sum_{i=1}^{M} |I| \cdot c_i \cdot k$ ejemplos para extraer conocimiento de ellas, mientras si se utilizaran las bases de datos refinadas analizaría $2 \cdot |M| \cdot |I|$ ejemplos. Puesto que en casos reales $c_i \cdot k >> 2 \Rightarrow 2 \cdot |M| \cdot |I| < \sum_{i=1}^{M} |I| \cdot c_i \cdot k$, el proceso de refinamiento propuesto es más eficiente que otras aproximaciones que trabajen directamente con datos crudos.

5.3.2. Minería de datos

Una vez que se ha reunido la información acerca del rendimiento y almacenado en las bases de datos refinadas DB, la fase de minería de datos extrae una colección de modelos de conocimiento. Estos modelos permitirán al agente coordinador adaptar el comportamiento de los agentes de optimización a diferentes instancias de P seleccionando los parámetros más adecuados y moviendo las soluciones candidatas entre los agentes.

La aproximación propuesta utiliza técnicas de minería de datos que extraen el conocimiento de DB y construyen el conjunto de modelos $T = \{t_{p_1^1}, \ldots, t_{p_1^n}, \ldots, t_{p_M^1}, \ldots, t_{p_M^n}\} \cup \{t^{\omega_1}, \ldots, t^{\omega_M}\}$, los cuales ayudan al agente coordinador en las siguientes tareas:

- Cada $t_{p_i^j} \in T$ proporciona el valor que debe tomar el parámetro j de m_i para resolver una instancia genérica de P del mejor modo posible.
- Por otro lado, cada t_{ω_i} proporciona conocimiento acerca del ranking de rendimientos computacionales mostrados por las metaheurísticas de M. En particular, el agente coordinador utiliza el modelo para obtener los pesos $\omega_i \in [0, 1]$, $i = 1, \ldots, |M|$, donde ω_i representa cuán apropiada es m_i para resolver una instancia dada de P.

5.3.2.1. Elección de la técnica de minería

Para poder aplicar esta fase, el primer paso necesario es escoger una técnica de minería de datos. Existe una amplia variedad de técnicas de minería de datos, basadas en diferentes

propuestas teóricas. Para este trabajo la búsqueda se centró en aquellas técnicas que de alguna manera permitieran tratamiento de datos imperfectos puesto que se esperaba tratar con ellos y que además dieran una salida fuzzy de tal manera que facilitara la obtención de las reglas fuzzy que modelan el coordinador. Además también se tuvo en cuenta que la técnica fuera fácilmente interpretable de forma que se pudiera extraer fácilmente el conocimiento.

Se han realizado algunos esfuerzos para incorporar incertidumbre e imprecisión en las fases de aprendizaje e inferencia de técnicas de minería de datos muy conocidas. Algunas técnicas que permiten el tratamiento de ejemplos con atributos desconocidos son las técnicas de construcción de árboles de regresión/clasificación. Además, en las técnicas de construcción de árboles también se permiten observaciones de atributos nominales que presentan valores inciertos desde el punto de vista probabilista, así como atributos continuos expresados mediante valores crisp. También se incorpora el tratamiento de atributos expresados mediante valores difusos. En general, se puede decir que una de las técnicas que en la actualidad realiza un tratamiento más completo de información imperfecta es la de árboles de decisión.

Los árboles además destacan por su sencillez, legibilidad e interpretabilidad. Y ésta fue la causa por la que finalmente se utilizó esta técnica de minería (para este caso es más interesante obtener modelos lo más claros posible para que sea más fácil comprenderlos y extraer su conocimiento).

Una vez se decidió que se usaría una técnica de árbol de decisión, el siguiente paso fue decidir que algoritmo de construcción de árboles sería más adecuada. Existen diferentes alternativas disponibles, de las cuales se consideraron CART [40], ID3 [223], C4.5 [225] y FDT [159]. Las dos primeras (CART e ID3) fueron descartadas puesto que no incorporan tratamiento de datos imperfectos. C4.5 aparece como respuesta a esto, ya que generaliza el algoritmo básico de construcción de árboles ID3 para tratar datos tanto nominales como continuos y que además permite el tratamiento de datos con un cierto tipo de imperfección. Entre estos tipos de imperfección, sin embargo, no se encuentran los conjuntos fuzzy (etiquetas lingüísticas). En este sentido, FDT, [159], mezcla la representación fuzzy y sus capacidades de razonamiento aproximado con la generación de árboles de decisión simbólicos uniendo las ventajas de ambos: el manejo de datos imperfectos junto con el alto grado de popularidad, comprensibilidad y fácil aplicación de esta técnica. Y por ello fue la técnica que finalmente se seleccionó. Por tanto, la notación de la metaheurística será $\text{KEPS}_{\mathcal{CP}\ \mathcal{T}}$, indicando el uso de modelos de árboles de decisión.

5.3.2.2. FDT

FDT es una técnica basada en árboles de decisión fuzzy que modifica el algoritmo básico ID3 en sus dos componentes, construcción del árbol e inferencia, para incorporar la representación fuzzy y el razonamiento aproximado. ID3 está basado en conjuntos clásicos, de forma que un ejemplo satisface solamente una de las condiciones de salida de un nodo y por lo tanto cae exactamente en un subnodo y consecuentemente en una sola hoja. En los árboles de decisión fuzzy, un ejemplo puede cumplir más de una condición, donde ahora las condiciones son restricciones fuzzy basadas en conjuntos fuzzy. Por esto, un ejemplo puede pertenecer a más de un hijo de un nodo y por lo tanto puede pertenecer o alcanzar más de un nodo hoja con algún grado en [0,1].

Contrucción del árbol e inferencia

El conjunto de ejemplos E a partir del cual se construye el árbol, está formado por ejemplos descritos por atributos que pueden ser nominales y numéricos y donde al menos habrá un atributo nominal que actúa como atributo clase. El algoritmo mediante el cual se construye el árbol fuzzy está basado en el algoritmo ID3 donde todos los atributos numéricos han sido discretizados mediante una partición fuzzy previamente obtenida mediante un algoritmo de discretización basado en un algoritmo genético [51].

En el Algoritmo 10, se muestra el algoritmo de construcción de un árbol fuzzy a partir de un conjunto de ejemplos.

La condición de parada está definida por la primera condición alcanzada de las siguientes: (1) nodo puro, (2) conjunto M vacío, (3) alcanzar el mínimo número de ejemplos permitido en un nodo.

Una vez construido el árbol lo usamos para inferir la clase desconocida de un nuevo ejemplo siguiendo el algoritmo 11.

Una característica del árbol fuzzy es que un ejemplo, en su recorrido por el árbol, puede alcanzar o activar más de una hoja. Por lo tanto, al hacer la inferencia se pueden usar múltiples estrategias. Algunas de ellas son:

- Considerar solamente la información proporcionada por la hoja activada con mayor valor por el ejemplo a clasificar.
- Considerar todas las hojas alcanzadas por el ejemplo y obtener la decisión final usando operadores de agregación como voto mayoritario, mínimo, máximo, promedio y versiones ponderadas de todos ellos.

Algoritmo 10: Pseudo-código del algoritmo de construcción del FDT.

Input: E: conjunto de ejemplos de entrenamiento, *Partición fuzzy*: partición de los atributos numéricos en conjuntos fuzzy
Output: t: Árbol fuzzy construido
begin
  Crear el nodo raíz, constituido por los ejemplos de E con valores;
  Construir M, el conjunto de atributos que describen los ejemplos de E y donde los atributos numéricos han sido discretizados según la *Partición Fuzzy*;
  Nodos_sin_expandir = {nodo raíz};
  while *no se satisfaga la condición de fin* **do**
    N = siguiente(Nodos_sin_expandir);
    foreach *atributo* $m \in M$ **do**
      Calcular la ganancia de información en M teniendo en cuenta la pertenencia de cada ejemplo e en el nodo N;
    end
    Elegir m_{part} el atributo con ganancia máxima;
    Dividir N en subnodos de acuerdo a las posibles salidas de m_{part};
    Añadir a Nodos_sin_expandir los subnodos de N;
    Eliminar m_{part} de M;
  end
end

Algoritmo 11: Pseudo-código del algoritmo de inferencia del FDT.

Input: t: FDT, *Partición fuzzy*: partición de los atributos numéricos en conjuntos fuzzy, e: Ejemplo que se desea inferir
Output: c: clase a la que pertenece el ejemplo
begin
  Recorrer el árbol desde el nodo raíz usando e;
  Obtener el conjunto de hojas alcanzadas por e;
  /* e puede ramificarse por más de una rama si en algún test basado en un atributo numérico, e pertenece a más de un elemento de la partición de dicho atributo con grado mayor que 0. También se puede ramificar por más de una rama si tiene al menos un valor desconocido en los atributos usados por el árbol como test. */
  foreach *n=hoja alcanzada por e* **do**
    Calcular el soporte para cada clase, obtenido como la suma de los valores de pertenencia de todos los ejemplos que alcanzaron dicha hoja durante la fase de aprendizaje y que pertenecen a dicha clase;
  end
  Obtener la decisión del árbol c a partir de la información proporcionada por el conjunto de hojas alcanzadas N_t y el valor con el que el ejemplo activa o alcanza cada hoja;
end

5.4. Clasificando la estrategia

Para terminar de describir la estrategia propuesta se clasificará la misma de acuerdo a las taxonomías expuestas en el capítulo 3.

Si consideramos la taxonomía de metaheurísticas híbridas propuesta por Talbi [241], la estrategia propuesta se incluiría en las de tipo HTH, que incluye aquellas metaheurísticas híbridas en las que metaheurísticas de distinto tipo llevan a cabo una búsqueda cooperativa.

Si se tiene en cuenta la taxonomía de metaheurísticas híbridas propuesta por Raidl en [226] se deben considerar diversas cuestiones. En primer lugar en respuesta a qué se hibrida, se presentan dos tipos de hibridación en este caso, por un lado se combinan metaheurísticas con metaheurísticas, pero también se combinan metaheurísticas con lógica fuzzy y con minería de datos en el caso del segundo esquema de cooperación. En cuanto al nivel de hibridación la estrategia se considera de alto nivel, el orden de ejecución es paralelo y la estrategia de colaboración es colaborativa.

Puesto que la estrategia que incluye el esquema de cooperación basado en conocimiento previo se considera un híbrido entre metaheurísticas y minería de datos, también se puede incluir en la taxonomía propuesta por Jourdan et al. en [161], de tal manera que la estrategia se puede definir como a priori si nos centramos en el tipo de conocimiento usado, puesto que éste es adquirido antes de la ejecución de la estrategia, su objetivo de cooperación es la mejor de la calidad de las soluciones y la parte de la metaheurística a la que se aplica el conocimiento son los parámetros, tanto de las metaheurísticas que cooperan como los de la misma estrategia cooperativa.

Capítulo 6

Estudio del comportamiento de la metaheurística KEPS

En la definición del sistema metaheurístico cooperativo adaptativo se dejaron muchos parámetros libres que requieren ser ajustados para obtener resultados adecuados. Estos son:

- ¿Qué metaheurísticas se deben utilizar? ¿Es mejor un sistema homogéneo o heterogéneo?
- ¿Qué arquitectura es más adecuada, la bifase o la monocapa?
- ¿Qué esquema de cooperación ofrece mejores resultados, el basado en memoria o el basado en conocimiento?
- ¿Cuánta cooperación es necesaria para obtener buenos resultados? En otras palabras ¿Qué valores deben tomar la frecuencia de intercambio de soluciones y el α *corte*?

En este capítulo se realizará un estudio de estos parámetros, con el objetivo de entender cómo influyen en el comportamiento del sistema. Para ello se realizará una serie de pruebas utilizando un problema de test, el problema de la mochila -{0,1}. Se debe tener en cuenta que el objetivo de este capítulo no es la obtención de resultados competitivos, sino el estudio de los parámetros, y por tanto se realizarán pruebas reducidas.

6.1. Un problema de test

Para realizar el ajuste del sistema se utilizará el conocido problema de la mochila-{0,1}. Este problema se puede definir formalmente como: Dado un conjunto de ítems, cada uno con un coste y un beneficio asociados, determinar un subconjunto tal que el coste total sea menor que un límite dado y que el beneficio total sea tan grande como sea posible. Su formalización

matemática es:

$$\begin{array}{ll} Max & \sum_{i=1}^{n} p_i \times x_i \\ s.a & \\ & \sum_{i=1}^{n} w_i \times x_i \leq C \\ & x_i \in \{0,1\}\, i = 1,...,n \end{array}$$

donde

- n es el número de ítems.
- x_i indica si un ítem i está incluido en la mochila o no.
- p_i es el beneficio asociado al ítem i.
- $w_i \in [0,\ldots,r]$ es el peso del ítem i.
- C es la capacidad máxima de la mochila.

Se asume que $w_i < C,\ \forall i$ y $\sum_{i=1}^{n} w_i > C$.

Este problema ha sido ampliamente estudiado en la literatura y se han propuesto varios algoritmos para su resolución. En [184] se puede encontrar un algoritmo basado en programación dinámica, que es uno de los más exitosos, mientras que en [23] se obtienen interesantes resultados, a partir de la combinación de los algoritmos más conocidos.

Para este problema se pueden construir instancias de dificultad variable modificando tanto el tamaño de las instancias como el tipo de correlación existente entre pesos y beneficios. A pesar de que el problema sea considerado uno de los problemas NP-duros "más fáciles", se pueden construir instancias "difíciles de resolver" que son consideradas un reto para la mayoría de algoritmos exactos actuales.

Un proceso que genera instancias difíciles es el presentado en [216]. Como Pisinger indica en [216]: "Hay dos direcciones a seguir cuando se construyen instancias difíciles: se pueden considerar instancias difíciles con coeficientes más grandes. Lo cual hará que los algoritmos basados en programación dinámica se ejecuten más lentamente, [...]. La segunda dirección que se puede seguir es construir instancias en las que los coeficientes sean de tamaño moderado, pero en los que los límites superiores actuales tengan un mal rendimiento." Por tanto, para construir y probar el sistema propuesto se utilizarán instancias con coeficientes o tamaño moderado, puesto que es más interesante centrarse en instancias intrínsecamente difíciles que en aquellas cuyo tamaño puede incrementar el coste computacional. En este apartado, se trabajará con un conjunto de instancias generadas por el Dr. David Pisinger con diferentes tamaños (el número de ítems que se pueden elegir) y con diferentes correlaciones entre pesos y beneficios. Estas instancias pertenecen a las siguientes cinco categorías, [216]:

- *Spanner*: Estas instancias se construyen de tal manera que todos sus ítems son múltiplos de un pequeño conjunto de ítems llamados *llave*. Esta llave puede ser generada usando cualquier distribución. En este caso se consideran tres distribuciones posibles:
  - *No correlacionadas (unc span)*: $w_i = U(1, R), p_i = U(1, R)$.
  - *Débilmente correlacionadas (wea span)*:$w_i = U(1, R)$, $p_i = U(w_i \quad \frac{R}{10}, w_i + \frac{R}{10})$ con $p_i \geq 1$.
  - *Fuertemente correlacionadas (str span)*: $w_i = U(1, R)$, $p_i = w_i + \frac{R}{10}$.
- *Profit ceiling (pceil)*: En estas instancias todos los beneficios son múltiplos de un parámetro dado d. En este caso $d = 3$: $w_i = U(1, R), p_i = d \lceil w_i/d \rceil$.
- *Circle*: Estas instancias son generadas de tal manera que los beneficios son una función de los pesos, teniendo su gráfica una representación elíptica ($d = \frac{2}{3}$): $w_i = U(1, R), p_i = d\sqrt{(4R^2 \quad (w \quad 2R)^2}$.

Los pesos w_i están distribuidos uniformemente en un intervalo con rango R. H instancias se generan de un modo aleatorio. La capacidad de cada instancia h, con $h = 1, \ldots, H$ se calcula como $c = \frac{h}{H+1} \sum_{i=1}^{n} w_i$. También se proporciona el óptimo para cada instancia.

6.2. Configuración de la metaheurística cooperativa

En esta sección se detallan diversos aspectos de la configuración del sistema que se utilizarán durante el estudio de parámetros.

6.2.1. Metaheurísticas usadas por los agentes de optimización

A continuación se mostrarán las distintas metaheurísticas individuales de las que podrán servirse los agentes de optimización. Estas son: Algoritmo Genético, Búsqueda Tabú, Optimización por Enjambre de Partículas y Temple Simulado. Todas ellas han sido implementadas salvo el Enjambre de Partículas que ha sido adaptado de la implementación propuesta en CILib [209, 210].

6.2.1.1. Implementación del Algoritmo Genético

El pseudo-código del Algoritmo Genético se presenta en el algoritmo 12.

Algoritmo 12: Pseudo-código del Algoritmo Genético.

Input: q: instancia del problema, os:operador de selección, pc: probabilidad de cruce, pm: probabilidad de mutación, ni: número de individuos que componen la población, r: proporción de individuos que pasan directamente a la siguiente población
Output: *solucion*: mejor solución encontrada
begin
  t = 0;
  inicializar(P(t));
  evaluar(P(t));
  while *no-finalizacion* **do**
    t = t + 1;
    Seleccionar P(t) desde P(t-1) usando os;
    P(t)=Cruzar(P(t)) con probabilidad p_c;
    P(t)=Mutar(P(t)) con probabilidad p_m;
    evaluar(P(t);
  end
  devolver mejor solucion;
end

De este algoritmo es necesario explicar los métodos de cruce y de mutación. El operador de cruce utilizado es el de cruce por un solo punto. El operador de mutación utilizado es cambio de un bit, en este caso la introducción/extracción de un elemento no presente/presente en la solución. El operador de selección utilizado es la selección por torneo.

El conjunto de parámetros P_{AG} que definen el comportamiento del Algoritmo Genético está compuesto por: el operador de selección, la probabilidad de cruce, la probabilidad de mutación, el número de individuos y la proporción de individuos que pasan directamente a la siguiente población.

6.2.1.2. Búsqueda Tabú

El algoritmo de Búsqueda Tabú aplicado se describe en el algoritmo 13.

Cabe destacar que como técnica de diversificación, la Búsqueda Tabú divide el espacio de búsqueda en una serie de regiones, y cuando es necesario diversificar genera una solución en la región menos visitada.

El conjunto de parámetros P_{BT} que definen el comportamiento de la Búsqueda Tabú está compuesto por: el tamaño de la lista tabú y el umbral de diversificación.

Algoritmo 13: Pseudo-código de la Búsqueda Tabú

Input: q: instancia del problema, tlt: tamaño de la lista tabú, ud: umbral de diversificación
Output: *solucion*: mejor solución encontrada
begin
  s_a = solucion-inicial(q);
  ListaTabu = { };
  while *no-finalizacion* **do**
    s_v = Mejorar(s_a,$\mathcal{N}(s_a)$,ListaTabu);
    $s_a = s_b$;
    Actualizar(ListaTabu);
    if *tiempo_sin_mejorar* > ud **then**
      s_a = diversifar(s_a);
    end
  end
  devolver s_a ;
end

6.2.1.3. Temple Simulado

El esquema del Temple Simulado utilizado se muestra en el algoritmo 14.

El conjunto de parámetros P_{TS} que definen el comportamiento del Temple Simulado está compuesto por: la función de enfriamiento utilizada, el número de iteraciones internas y el número de soluciones que se generan antes de seleccionar una para comprobar si se toma como solución actual.

Como funciones de enfriamiento se utilizan las siguientes:

- Tangente hiperbólica:

$$t_{max} * (1 \quad \frac{\frac{e^k \quad e^{-k}}{2}}{\frac{e^k + e^{-k}}{2}})$$

- Coseno hiperbólico:

$$\frac{t_{max}}{\frac{e^{\frac{10 \cdot k}{t_{max}}} + e^{-\frac{10 \cdot k}{t_{max}}}}{2}}$$

- Raíz cuadrada:

$$\frac{t_{max}}{\sqrt{k+1}}$$

- Exponencial:

$$e^{\ k} \cdot t_{max}$$

Algoritmo 14: Pseudo-código del Temple Simulado

Input: q: instancia del problema, it_{max}: número de iteraciones internas máximas, $f_{enfriamiento}$: función de enfriamiento, n_{max}: vecinos máximos
Output: *solucion*: mejor solución encontrada
begin
  k=0;
  t_0=T;
  s_a = solucion-inicial(q);
  while *no-finalizacion* **do**
    while $it_{internas} < it_{max}$ **do**
      $it_{internas} = it_{internas} + 1$;
      while $n_{vecinos} < n_{max}$ **do**
        $n_{vecinos} = n_{vecinos} + 1$;
        s_v = GenerarVecino(s_a);
        if $(f(s_v) \quad f(s_a)) > 0$ **then**
          $s_a = s_v$;
        else
          if $rand(0,1) \leq e^{(f(s_v) \quad f(s_a))/t_k}$ **then**
            $s_a = s_v$;
          end
        end
      end
      k=k+1;
      t_k=$f_{enfriamiento}(t_{k \ 1})$;
    end
  end
  devolver s_a;
end

6.2.1.4. Optimización por Enjambre de Partículas

El esquema de esta estrategia se muestra en el algoritmo 15.

El conjunto de parámetros P_{OEP} que definen el comportamiento de la técnica está compuesto por: el número de partículas, la topología que las une, w, parámetro que controla la velocidad, r_p parámetro que controla la memoria histórica, r_g parámetro que controla la memoria global.

Algoritmo 15: Pseudo-código de la Optimización por Enjambre de Partículas.

Input: q: instancia del problema,$topografia$:topografía de unión de partículas, ω: inercia, r_p: control de memoria histórica,r_g: control de memoria global
Output: *solucion*: mejor solución encontrada
begin
  foreach *particula* **do**
    inicPartic($topografia$);
  end
  while *no-finalizacion* **do**
    foreach *particula p* **do**
      fitn = calcFitness(p);
      if *fitn es mejor que la mejor partícula pBest* **then**
        pBest = p;
      end
    end
    gBest = particulaMejor(pBest);
    foreach *particula i* **do**
      $v_i = \omega v_i + rand(0,1) \cdot r_p \cdot (pBest_i \quad x_i) + rand(0,1) \cdot r_g \cdot (gBest_i \quad x_i)$;
      $x_i = x_i + v_i$;
    end
  end
end

6.2.2. Aplicación del proceso de aprendizaje para el esquema de cooperación basado en conocimiento

En este apartado se muestra la aplicación al problema de la mochila utilizando en el esquema de cooperación basado en conocimiento previo.

6.2.2.1. Preparación de datos

Cuando se aplica el proceso de extracción del conocimiento, el primer paso es reunir la información obtenida de la aplicación independiente de las metaheurísticas para resolver instancias de entrenamiento del problema. En el caso del problema de la mochila se generaron 100 instancias de entrenamiento con $R = 10^3$, cuatro tamaños de problema, $n = (500, 1000, 1500, 2000)$, y por cada n y cada tipo de instancia, 5 instancias con h generado aleatoriamente entre $[1, 100]$.

Cada una de las instancias se resolvió con cada una de las metaheurísticas que se quieren incluir en el sistema (el Algoritmo Genético, la Búsqueda Tabú, el Temple Simulado

y la Optimización por Enjambre de Partículas), tomando los siguientes valores para sus parámetros:

- Para el Algoritmo Genético su conjunto de parámetros (P_{AG}) tomó los siguientes valores:
  - Operador de selección: selección por torneo.
  - Probabilidad de cruce: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, y 0.9.
  - Probabilidad de mutación: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, y 0.9.
  - Número de individuos: 20, 40, 60 y 80.
  - Proporción de individuos que pasan directamente: 0.01, 0.1 y 0.2.
- Para la Busqueda Tabú su conjunto de parámetros (P_{BT}) tomó los siguientes:
  - Tamaño de la lista tabú: 5, 10, 50, 100, 200 y 500.
  - Umbral de diversificación: 1, 2, 10, 50 y 100.
- En el caso del Temple Simulado su conjunto de parámetros (P_{TS}) tomó los siguientes:
  - Función de enfriamiento: Tangente hiperbólica, coseno hiperbólico, raíz cuadrada y exponencial.
  - Número de iteraciones internas: 10, 100, 500, 1000 y 10000.
  - Número de soluciones generadas antes de la selección: 1 2 3 5 y 10.
- Por último para la Optimización por Enjambre de Partículas, los parámetros incluidos en P_{OEP} tomaron los siguientes valores:
  - Número de partículas: 20, 30 y 40.
  - Topología: Mejor local considerando 4 vecinos, mejor local considerando 6 vecinos, mejor global y Von Neumann.
  - ω: 0.6, 0.729844, y 0.8.
  - r_p: 1.1, 1.2, 1.49, y 1.6.
  - r_g: 1.4, 1.49, 1.6 y 1.8.

Con la información generada por estas ejecuciones se obtuvo el conjunto de bases de datos crudas cuyo posterior refinamiento proporcionó el conjunto de bases de datos refinadas BD. De estas tenemos una base de datos para calcular el peso de cada metaheurística y una base de datos para calcular los mejores parámetros para cada metaheurística. Cada una de estas bases de datos contiene los siguientes atributos:

- Una descripción de la instancia del problema, incluyendo: el número de objetos disponibles (tamaño); la proporción máxima, mínima y media de $w_i/capacidad$ de los objetos

disponibles ($max(w_i/cap)$, $min(w_i/cap)$, $med(w_i/cap)$); las proporciones máxima, mínima y media de p_i/w_i de los objetos disponibles ($max(p_i/w_i)$,$min(p_i/w_i)$, $med(p_i/w_i)$), la desviación típica de p_i/w_i ($\sigma(p_i/w_i)$), el número de valores distintos de p_i/w_i (nPendientes) y tres estimaciones del tipo de instancia que distinguen, entre spanner, profit ceiling y circle (esSpan, esPCeil, esCircle).

- En caso de que sea una base de datos refinada de parámetros, los valores de los mejores parámetros para la instancia descrita.
- En caso de que sea una base de datos de pesos, los valores más adecuados de pesos para la instancia descrita.

Se pueden observar extractos de cada una de ellas en las tablas 6.1 y 6.2.

Tabla 6.1: Parte de la base de datos de parámetros de la Búsqueda Tabú

Descripción de la instancia

Size	$max(w_i/cap)$	$min(w_i/cap)$	$med(w_i/cap)$
2000	2.19E-03	8.00E-05	1.56E-03
2000	2.24E-03	8.10E-05	1.58E-03
...	...	...	...

$max(p_i/w_i)$	$min(p_i/w_i)$	$med(p_i/w_i)$	$\sigma(p_i/w_i)$
8.67E-01	2.38E-02	5.46E-01	1.38E-01
8.67E-01	2.38E-02	5.43E-01	1.38E-01
...	...	...	...

nPendientes	*esSpan*	*esPCeil*	*esCircle*
869	false	false	true
856	false	false	true
...	...	...	...

Información de parámetros

lli	*ui*
100	10
10	10
...	...

Tabla 6.2: Parte de la base de datos de pesos de la Búsqueda Tabú

Descripción de la instancia

Size	$max(w_i/cap)$	$min(w_i/cap)$	$med(w_i/cap)$
2000	2.19E-03	8.00E-05	1.56E-03
2000	2.24E-03	8.10E-05	1.58E-03
...	...	...	...

$max(p_i/w_i)$	$min(p_i/w_i)$	$med(p_i/w_i)$	$\sigma(p_i/w_i)$
8.67E-01	2.38E-02	5.46E-01	1.38E-01
8.67E-01	2.38E-02	5.43E-01	1.38E-01
...	...	...	...

nPendientes	*esSpan*	*esPCeil*	*esCircle*
869	false	false	true
856	false	false	true
...	...	...	...

Información de pesos

peso
0.45
0.16
...

6.2.2.2. Minería de datos

Después de obtener el conjunto de bases de datos refinadas BD, se puede aplicar la fase de minería de datos y obtener los árboles de decisión fuzzy que configuran al coordinador. Pero antes de poder aplicar la técnica de árboles de decisión fuzzy escogida es necesario particionar todos los atributos numéricos en distintos conjuntos fuzzy, lo cual se hizo utilizando el algoritmo propuesto en [51]. En el caso de la mochila, se obtendrán 4 árboles de pesos y 15 de parámetros. Un ejemplo de estos árboles se puede ver en la figura 6.1.

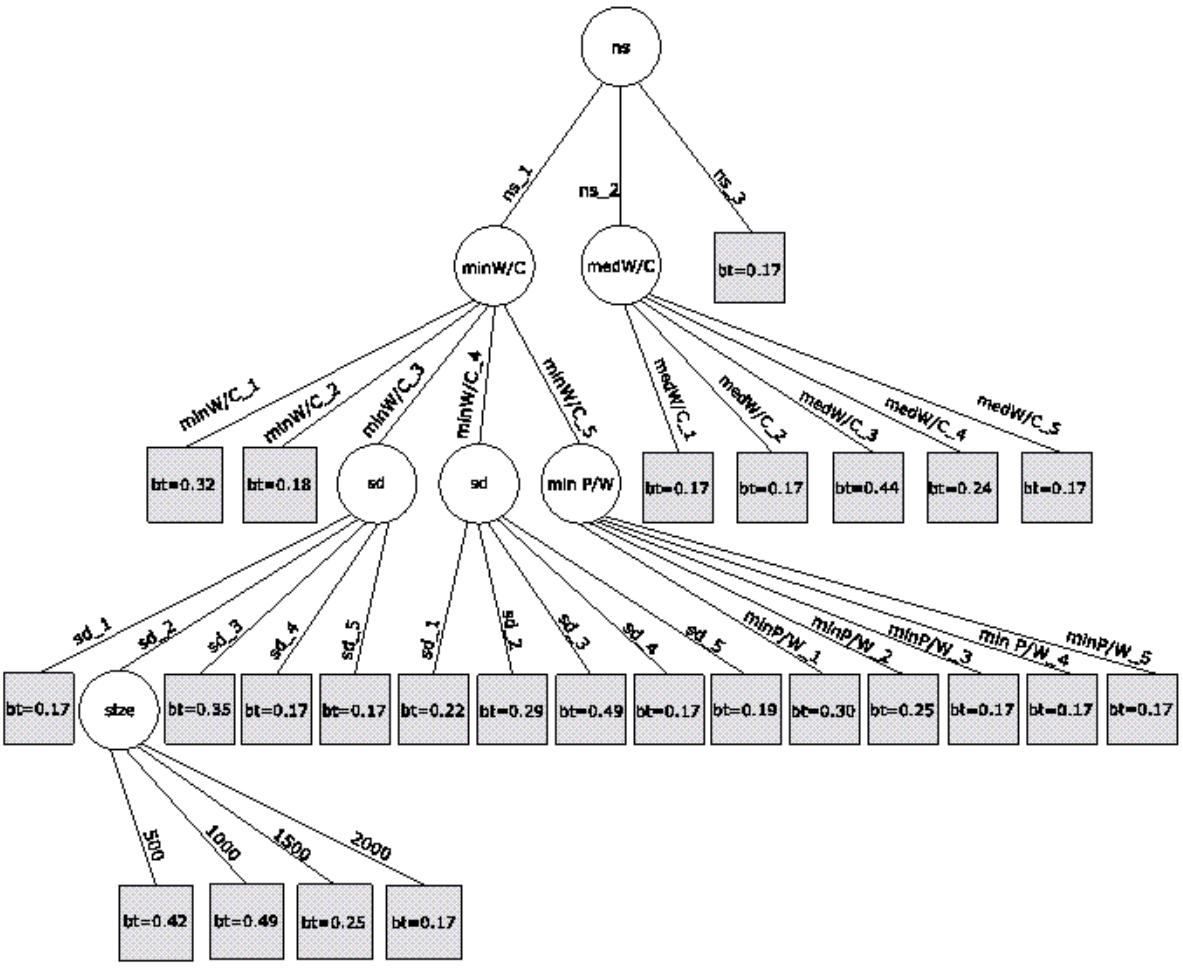


Figura 6.1: Ejemplo de árbol de pesos.

6.2.3. Modo de ejecución

Cuando se prueban metaheurísticas cooperativas, se puede optar por utilizar esquemas paralelos si el tiempo se considera importante, o se puede simular el paralelismo en un

ordenador mono-procesador. Esta última es la estrategia que hemos seguido. Se construye un array de agentes de optimización y se ejecutan siguiendo una aproximación round-robin. De esta manera los agentes son ejecutados durante cierto número de evaluaciones de la función objetivo. El número de evaluaciones es aleatorio en un intervalo dado, y después de este periodo de tiempo, el intercambio de información es llevado a cabo. Después de la ejecución de todos los agentes de optimización, se ejecuta el agente coordinador para que lleve a cabo sus funciones correspondientes. Estos pasos se repiten hasta que se alcance la condición de parada, dada en términos de evaluaciones de la función objetivo.

Por último, en cada sección y una vez que se hayan mostrado los resultados, se realiza un análisis de los métodos usando técnicas estadísticas. Siguiendo la metodología propuesta en [123], que utiliza tests no paramétricos. En particular se utiliza el test de Wilcoxon para comparar dos métodos. Este test es un procedimiento no paramétrico para realizar comparaciones pareadas entre dos algoritmos. Es el análogo no paramétrico del t-test pareado. Por tanto es un test paread que trata de detectar diferencias significativas entre dos aproximaciones. Cuando se comparan varios métodos (más de dos), se utiliza el test de Friedman y el método Benjamin-Hochberger como test post-hoc. Este último método es más potente que el test Bonferroni-Dunn, el test Holm y el método Hochberger. El test de Friedman es el equivalente no paramétrico al ANOVA. Como hipótesis nula, considera que todos los algoritmos son equivalentes, de tal manera que el rechazo de esta hipótesis implica la existencia de diferencias en el rendimiento de los algoritmos estudiados. Después de esto se puede usar el test post-hoc para discernir si el algoritmo propuesto presenta diferencias estadísticas con respecto al resto de métodos de la comparación.

6.3. Análisis de la influencia de las metaheurísticas

En la introducción del capítulo se introdujeron varias cuestiones que surgen a la hora de diseñar el sistema. En esta sección se pretende responder a la primera, ¿Qué metaheurísticas se deben utilizar? y ¿Es mejor un sistema homogéneo o heterogéneo? Para ello se realizará un estudio de cómo afectan a la metaheurística cooperativa las distintas metaheurísticas que la componen.

En esta sección se compararán las metaheurísticas aisladas con metaheurísticas cooperativas homogéneas, es decir compuestas por un solo tipo de metaheurística y con una metaheurística cooperativa heterogénea. En particular se usarán las siguientes metaheurísticas:

- AG: Algoritmo Genético aislado.
- BT: Búsqueda Tabú aislado.

- OEP: Optimización de Enjambre de Partículas aislado.
- TS: Temple Simulado aislado.
- H-AG: metaheurística cooperativa compuesta de cuatro AGs.
- H-BT: metaheurística cooperativa compuesta de cuatro BTs.
- H-OEP: metaheurística cooperativa compuesta de cuatro OEPs.
- H-TS: metaheurística cooperativa compuesta de cuatro TSs.
- Heterogéneo: metaheurística cooperativa compuesta de un AG, BT, OEP y TS.

Para estas pruebas se utilizaron 20 instancias (1 por tipo y tamaño) distintas de las que se utilizaron en el aprendizaje. Cada estrategia fue ejecutada 10 veces usando 100000 evaluaciones de la función objetivo. En el caso de las metaheurísticas cooperativas la frecuencia de intercambio viene determinada por el intervalo $[5000, 5500]$ y se utiliza un α $corte$ de 0.5. Los resultados mostrados corresponden al error medio obtenido calculado como $error = 100 \times \frac{valor obtenido \quad optimo}{optimo}$.

En la Fig. 6.2 se muestra un resumen de los resultados obtenidos por todas las metaheurísticas.

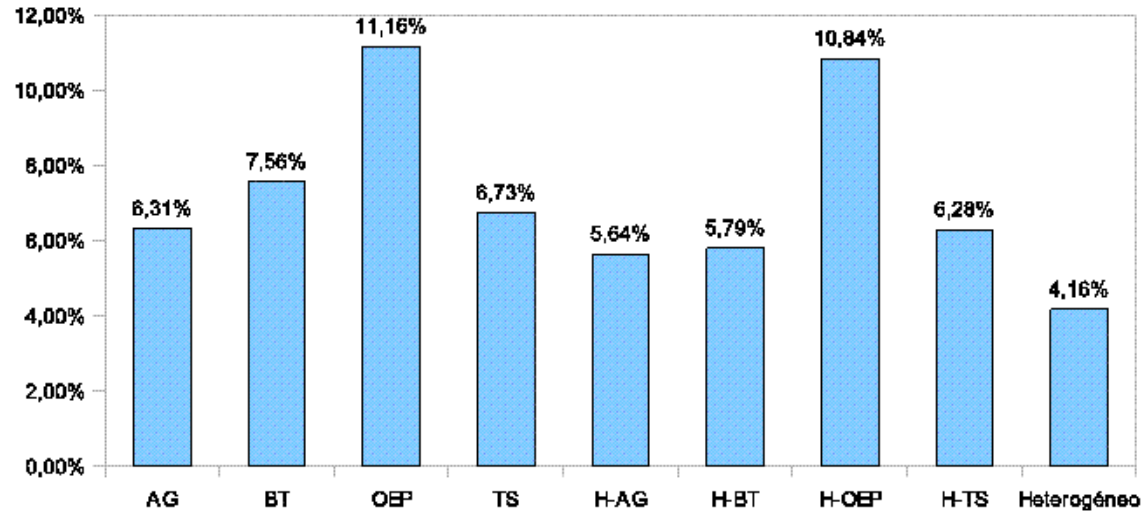


Figura 6.2: Resumen de los resultados obtenidos por las metaheurísticas

En ella se puede observar cómo las metaheurísticas cooperativas obtienen siempre un mejor comportamiento que las individuales, siendo la mejora del rendimiento bastante importante en los casos del Algoritmo Genético y la Búsqueda Tabú y, por contra, muy pequeña en los casos del Temple Simulado y la Optimización por Enjambre de Partículas. Por otro lado, se puede

observar cómo la estrategia cooperativa heterogénea es capaz de obtener unos resultados tan buenos como los obtenidos por la mejor estrategia homogénea e incluso un poco mejores.

Los resultados obtenidos se muestran de modo más detallado en las Tablas 6.3 y 6.4 que indican los resultados de las metaheurísticas individuales y cooperativas respectivamente. En estas tablas se muestra para cada tamaño de instancia y para cada tipo el error medio obtenido y como subíndice su desviación típica.

Tabla 6.3: Resultados de las metaheurísticas individuales

tamaño	tipo	AG	BT	OEP	TS
500	unc span	$9{,}57_{1{,}03}$	$7{,}29_{1{,}29}$	$30{,}48_{0{,}99}$	$8{,}06_{0{,}53}$
	wea span	$2{,}47_{0{,}15}$	$5{,}14_{0{,}24}$	$5{,}93_{0{,}43}$	$5{,}33_{0{,}18}$
	str span	$1{,}04_{0{,}04}$	$1{,}21_{0{,}04}$	$1{,}55_{0{,}07}$	$1{,}33_{0{,}03}$
	pceil	$0{,}12_{0{,}01}$	$0{,}10_{0{,}00}$	$0{,}22_{0{,}07}$	$0{,}13_{0{,}00}$
	circle	$9{,}87_{0{,}52}$	$6{,}95_{0{,}55}$	$17{,}21_{0{,}64}$	$16{,}58_{0{,}65}$
1000	unc span	$13{,}61_{1{,}13}$	$19{,}79_{1{,}49}$	$30{,}20_{0{,}53}$	$8{,}84_{0{,}38}$
	wea span	$3{,}14_{0{,}13}$	$4{,}99_{0{,}05}$	$5{,}44_{0{,}25}$	$5{,}25_{0{,}18}$
	str span	$1{,}16_{0{,}03}$	$1{,}32_{0{,}02}$	$1{,}56_{0{,}11}$	$1{,}40_{0{,}03}$
	pceil	$0{,}13_{0{,}00}$	$0{,}11_{0{,}00}$	$0{,}23_{0{,}07}$	$0{,}14_{0{,}00}$
	circle	$11{,}86_{0{,}27}$	$11{,}23_{0{,}73}$	$17{,}07_{0{,}54}$	$17{,}27_{0{,}28}$
1500	unc span	$17{,}25_{0{,}71}$	$24{,}82_{1{,}02}$	$31{,}68_{0{,}61}$	$10{,}07_{0{,}45}$
	str span	$3{,}63_{0{,}13}$	$5{,}21_{0{,}14}$	$5{,}60_{0{,}19}$	$5{,}54_{0{,}16}$
	wea span	$1{,}26_{0{,}02}$	$1{,}40_{0{,}02}$	$1{,}56_{0{,}06}$	$1{,}46_{0{,}02}$
	pceil	$0{,}14_{0{,}00}$	$0{,}13_{0{,}00}$	$0{,}19_{0{,}02}$	$0{,}15_{0{,}00}$
	circle	$13{,}06_{0{,}33}$	$13{,}47_{0{,}44}$	$17{,}37_{0{,}31}$	$17{,}87_{0{,}39}$
2000	unc span	$18{,}93_{0{,}96}$	$26{,}80_{0{,}56}$	$31{,}95_{0{,}38}$	$10{,}21_{0{,}31}$
	wea span	$3{,}81_{0{,}13}$	$5{,}17_{0{,}05}$	$5{,}59_{0{,}17}$	$5{,}47_{0{,}12}$
	str span	$1{,}29_{0{,}03}$	$1{,}43_{0{,}01}$	$1{,}55_{0{,}05}$	$1{,}47_{0{,}03}$
	pceil	$0{,}14_{0{,}00}$	$0{,}13_{0{,}00}$	$0{,}19_{0{,}02}$	$0{,}15_{0{,}00}$
	circle	$13{,}68_{0{,}33}$	$14{,}41_{0{,}43}$	$17{,}64_{0{,}28}$	$17{,}85_{0{,}30}$

El análisis detallado de estas tablas nos proporciona información interesante. Inicialmente, utilizando el test estadístico de Friedman, se observa que existen diferencias significativas entre las metaheurísticas utilizadas con una certeza del 99.99 %. Realizando el análisis post-hoc, obtenemos los siguientes resultados: si nos centramos en las metaheurísticas aisladas podemos observar cómo el Algoritmo Genético es aquél que obtiene un mejor comportamiento, seguido por la Búsqueda Tabú, después el Temple Simulado y por último la Optimización por Enjambre de Partículas, este ranking está corroborado por los tests estadísticos antes mencionados. Es interesante comprobar como no siempre el algoritmo genético es la mejor metaheurística, sino que en varias ocasiones se ve superada por otras metaheurísticas. Este hecho justifica la utilización de metaheurísticas cooperativas heterogéneas, puesto que son susceptibles de adaptarse a la metaheurística que obtenga un mejor comportamiento. Entre

Tabla 6.4: Resultados de las metaheurísticas cooperativas

tamaño	tipo	Homogéneo				
		AG	BT	OEP	TS	Heterogéneo
500	unc span	$6{,}03_{D,64}$	$11{,}24_{D,65}$	$29{,}56_{D,62}$	$7{,}87_{D,61}$	$4{,}58_{D,69}$
	wea span	$2{,}25_{D,10}$	$4{,}53_{D,19}$	$5{,}60_{D,35}$	$5{,}17_{D,31}$	$1{,}44_{D,24}$
	str span	$1{,}00_{D,04}$	$1{,}14_{D,02}$	$1{,}43_{D,07}$	$1{,}29_{D,04}$	$1{,}01_{D,04}$
	pceil	$0{,}16_{D,00}$	$0{,}10_{D,00}$	$0{,}16_{D,02}$	$0{,}13_{D,00}$	$0{,}11_{D,00}$
	circle	$8{,}86_{D,44}$	$8{,}86_{D,40}$	$16{,}41_{D,59}$	$15{,}85_{D,46}$	$7{,}62_{D,62}$
1000	unc span	$11{,}41_{D,60}$	$13{,}81_{D,57}$	$29{,}75_{I,04}$	$6{,}20_{D,43}$	$6{,}28_{D,96}$
	wea span	$2{,}95_{D,16}$	$4{,}85_{D,05}$	$5{,}22_{D,18}$	$4{,}93_{D,13}$	$2{,}29_{D,19}$
	str span	$1{,}12_{D,02}$	$1{,}24_{D,02}$	$1{,}46_{D,03}$	$1{,}36_{D,03}$	$1{,}13_{D,03}$
	pceil	$0{,}13_{D,00}$	$0{,}11_{D,00}$	$0{,}17_{D,03}$	$0{,}14_{D,00}$	$0{,}12_{D,00}$
	circle	$11{,}27_{D,31}$	$10{,}60_{D,24}$	$16{,}86_{D,40}$	$16{,}54_{D,31}$	$10{,}31_{D,56}$
1500	unc span	$14{,}94_{D,50}$	$13{,}35_{D,55}$	$30{,}71_{D,81}$	$8{,}54_{D,36}$	$8{,}51_{D,77}$
	str span	$3{,}58_{D,08}$	$5{,}01_{D,09}$	$5{,}53_{D,11}$	$5{,}23_{D,12}$	$2{,}69_{D,15}$
	wea span	$1{,}23_{D,03}$	$1{,}30_{D,02}$	$1{,}49_{D,04}$	$1{,}43_{D,02}$	$1{,}22_{D,03}$
	pceil	$0{,}14_{D,00}$	$0{,}12_{D,00}$	$0{,}17_{D,01}$	$0{,}14_{D,00}$	$0{,}12_{D,00}$
	circle	$12{,}20_{D,29}$	$10{,}50_{D,12}$	$17{,}11_{D,29}$	$16{,}83_{D,36}$	$10{,}60_{D,42}$
2000	unc span	$17{,}28_{D,76}$	$12{,}43_{D,72}$	$30{,}78_{D,45}$	$9{,}99_{D,34}$	$10{,}27_{D,60}$
	wea span	$3{,}75_{D,08}$	$4{,}99_{D,04}$	$5{,}49_{D,09}$	$5{,}20_{D,16}$	$3{,}02_{D,21}$
	str span	$1{,}26_{D,01}$	$1{,}32_{D,03}$	$1{,}49_{D,04}$	$1{,}44_{D,03}$	$1{,}25_{D,03}$
	pceil	$0{,}14_{D,00}$	$0{,}12_{D,00}$	$0{,}17_{D,02}$	$0{,}15_{D,00}$	$0{,}12_{D,00}$
	circle	$13{,}07_{D,26}$	$10{,}18_{D,21}$	$17{,}22_{D,17}$	$17{,}23_{D,28}$	$10{,}49_{D,40}$

las metaheurísticas cooperativas homogéneas se el orden es similar, salvo por el hecho de que no se aprecian diferencias significativas entre H-GA y H-BT, aunque sí respecto a estos dos y el resto de metaheurísticas. Por último la que obtiene mejores resultados es la metaheurística cooperativa heterogénea, corroborado por los tests estadísticos. También es interesante comprobar cómo en casi todas las instancias analizadas las metaheurísticas cooperativas mejoran los resultados obtenidos por sus homólogas independientes.

A continuación realizaremos un análisis más exhaustivo de los datos valiéndonos de gráficos explicativos. En estos gráficos se confrontan las metaheurísticas por parejas. Cada instancia se representa por un punto (x, y) donde $x(y)$ es la media de error normalizado obtenido por la estrategia nombrada en el eje $X(Y)$. Se utiliza un círculo para mostrar cuando las diferencias entre ambos algoritmos no son significativos (normalmente cuando el punto está muy cercano a la diagonal), en otro caso se utiliza un triángulo. Si el punto se encuentra por encima de la diagonal, la metaheurística en X es mejor que la que se encuentra en Y y viceversa.

En la Figura 6.3 se muestra una comparación entre el Algoritmo Genético individual y el cooperativo homogéneo, como puede observarse la superioridad de la metaheurística cooperativa es manifiesta, puesto que casi todos los puntos están por encima de la diagonal y

además existen diferencias estadísticamente significativas.

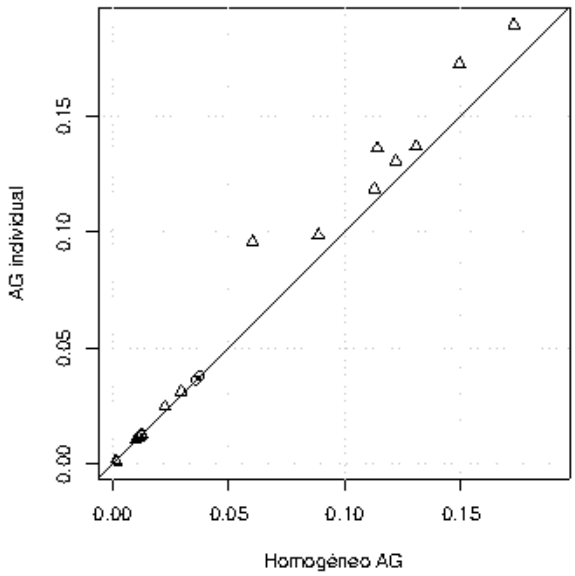


Figura 6.3: Comparativa entre AG y H-AG

En la figura 6.4 se comparan la Búsqueda Tabú individual con la cooperativa homogénea.

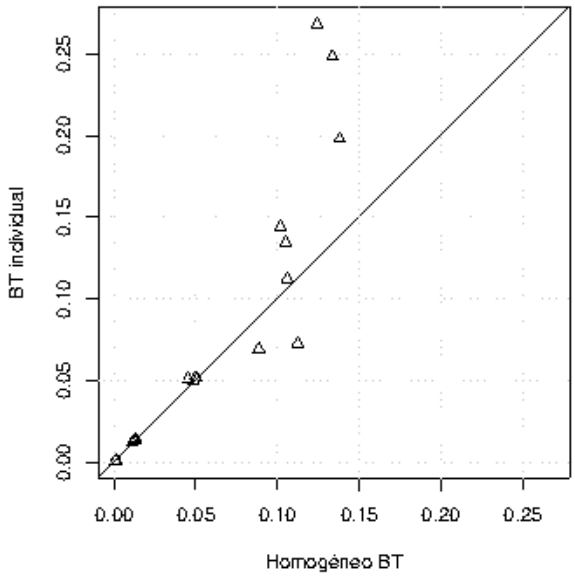


Figura 6.4: Comparativa entre BT y H-BT

En este caso la superioridad de la metaheurística cooperativa también es clara, aunque en algunos casos la metaheurística individual obtiene mejores resultados.

Para comparar la Optimización por Enjambre de Partículas individual con la cooperativa homogénea se utiliza la Figura 6.5.

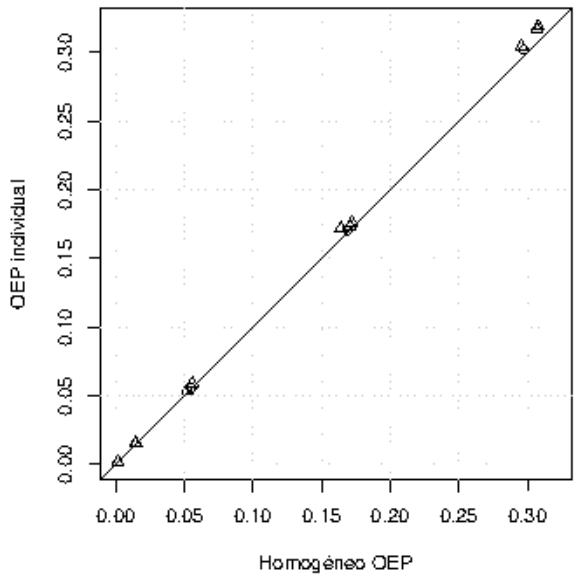


Figura 6.5: Comparativa entre OEP y H-OEP

En este caso los superioridad no es tan clara. A pesar de ello, las diferencias que se muestran también son significativas, obteniéndose mejoras para todas las instancias, aunque no tan grandes como en los casos anteriores. Esto puede ser debido al mal comportamiento que muestra la metaheurística en general.

La última comparación entre metaheurísticas individuales y sus respectivas metaheurísticas cooperativas homogéneas corresponde al Temple Simulado, y se muestra en la Figura 6.6. Este caso es similar al anterior, puesto que se obtienen mejoras significativas en todas las instancias, pero no tan grandes como en los casos anteriores.

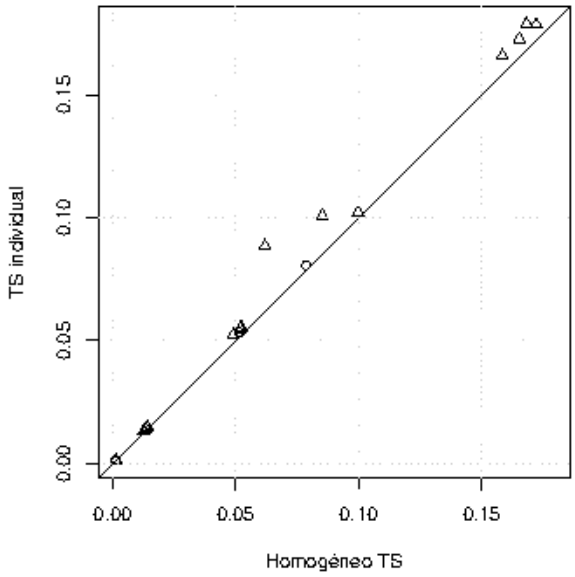


Figura 6.6: Comparativa entre TS y H-TS

A continuación se compararán los rendimientos de las distintas metaheurísticas cooperativas. En las figuras 6.7, 6.8 y 6.9 se comparan los resultados de H-AG con el resto de las metaheurísticas cooperativas. Del análisis de estos gráficos se desprende la superioridad de H-AG que obtiene resultados estadísticamente mejores en la mayoría de instancias.

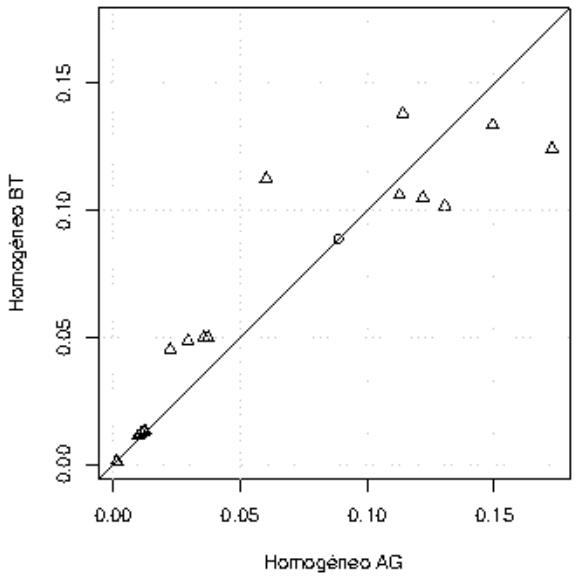


Figura 6.7: Comparativa entre H-AG y H-BT

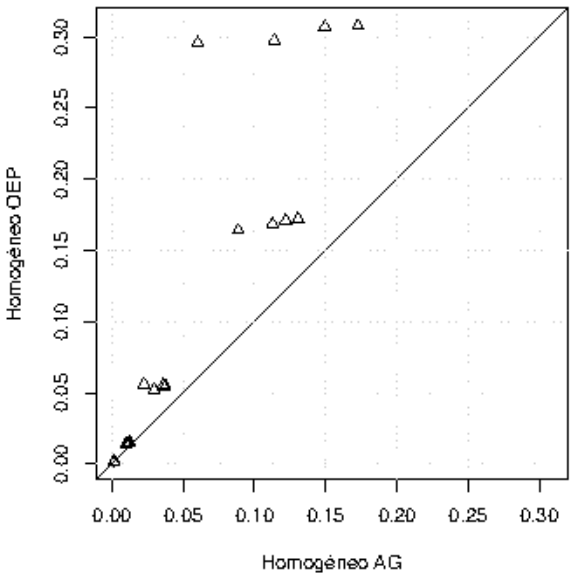


Figura 6.8: Comparativa entre H-AG y H-OEP

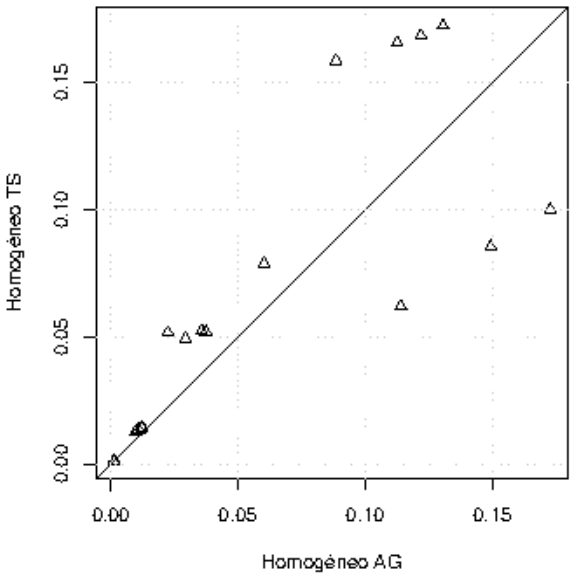


Figura 6.9: Comparativa entre H-AG y H-TS

En las figuras 6.10 y 6.11 se muestran las comparaciones entre H-BT y el resto de metaheurísticas cooperativas homogéneas. Con respecto a H-OEP se puede observar como H-BT obtiene claramente mejores resultados. Por otro lado si la comparamos con H-TS la mejoría no es tan clara puesto que existen algunas instancias para las que H-TS obtiene mejores resultados, sin embargo en la mayoría de los casos es H-BT la que obtiene los mejores.

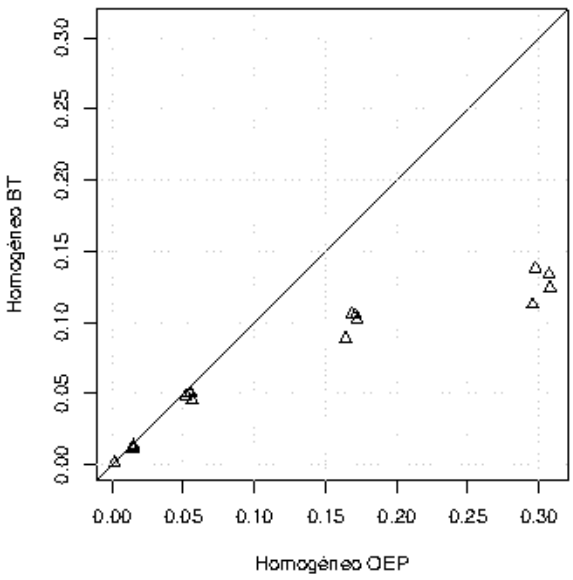


Figura 6.10: Comparativa entre H-BT y H-OEP

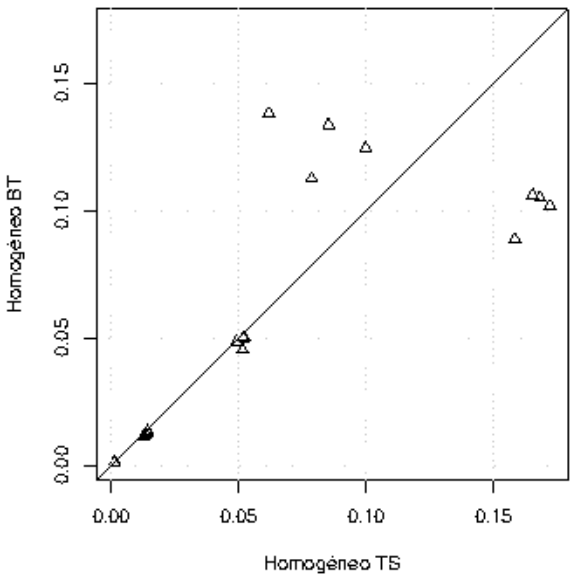


Figura 6.11: Comparativa entre H-BT y H-TS

Por último en la figura 6.12 se comparan H-OEP y H-TS. Como puede observarse, en la mayoría de instancias el rendimiento es similar salvo para algunas en las que los resultados obtenidos por H-TS son mucho mejores que los obtenidos por H-OEP.

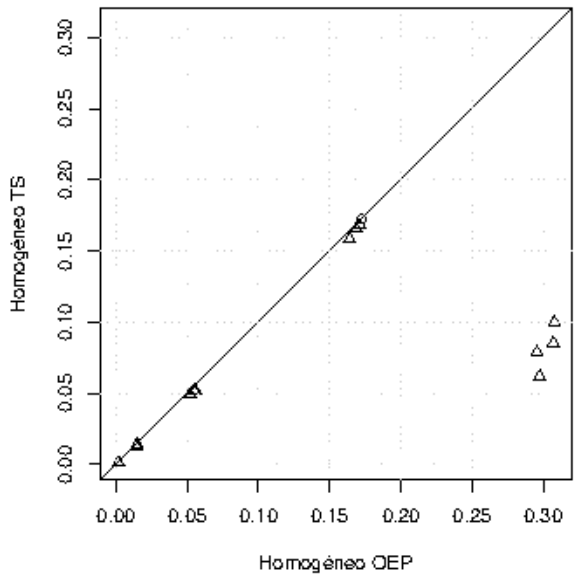


Figura 6.12: Comparativa entre H-OEP y H-TS

Finalmente en las figuras 6.13, 6.14, 6.15 y 6.16 se compara la estrategia heterogénea con todas las homogéneas. La superioridad de la metaheurística cooperativa heterogénea es manifiesta en todos los casos, obtiendo resultados mucho mejores en la mayoría de instancias, salvo si se compara con H-AG en cuyo caso el comportamiento es similar obteniéndose casi tantos puntos por encima de la diagonal como por debajo, sin embargo cuando la metaheurística cooperativa heterogénea obtiene mejores resultados las diferencias son mayores.

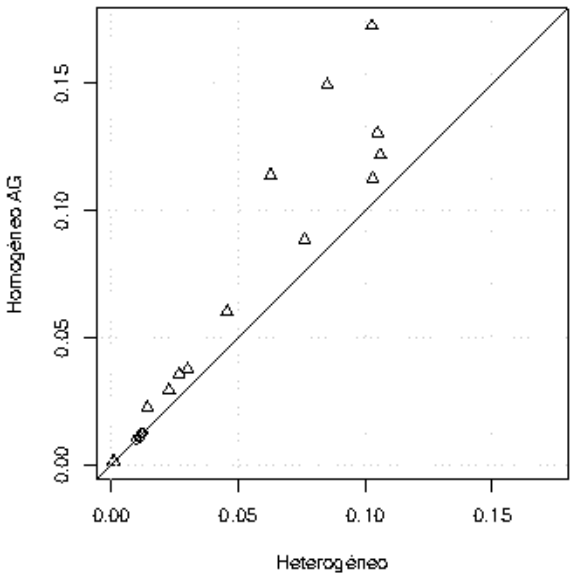


Figura 6.13: Comparativa entre Heterogénea y H-AG

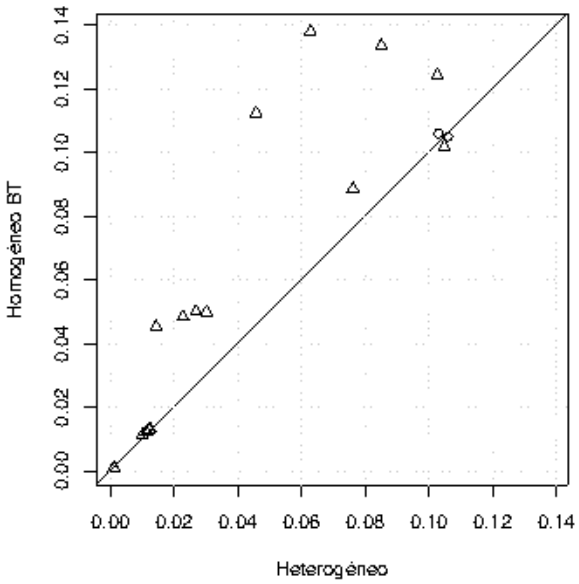


Figura 6.14: Comparativa entre Heterogénea y H-BT

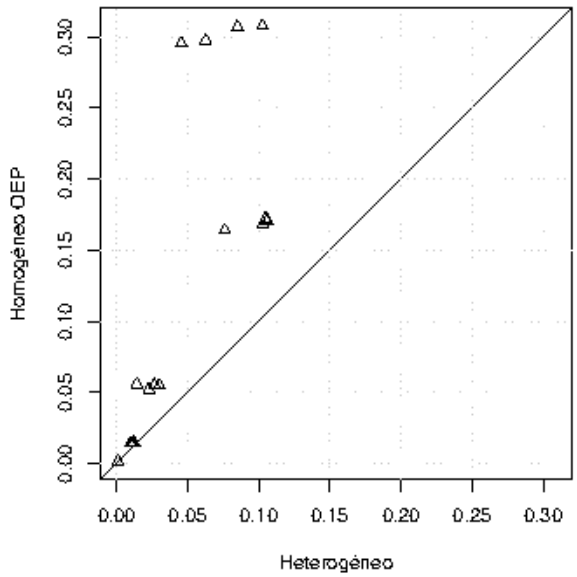


Figura 6.15: Comparativa entre Heterogénea y H-OEP

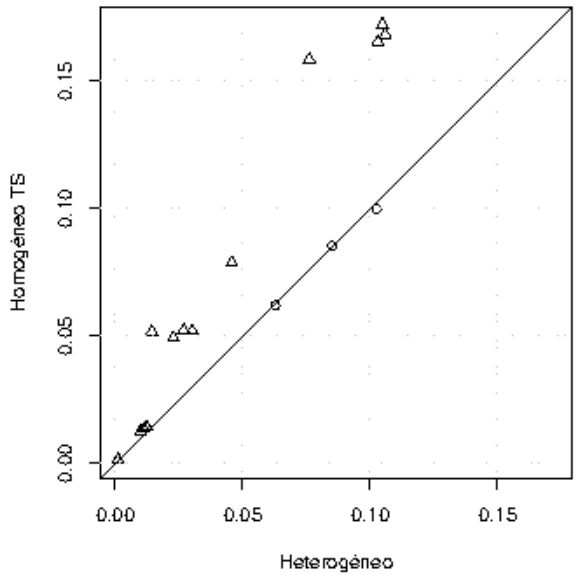


Figura 6.16: Comparativa entre Heterogénea y H-TS

6.4. Análisis de la influencia de la arquitectura utilizada

En esta sección se comprobará como influye el uso de las dos arquitecturas propuestas, tanto la bicapa KEPS^2, dividida en dos fases secuenciales en las que se ejecutan respectivamente metaheurísticas basadas en poblaciones y en trayectorias, como la monocapa o KEPS^1, en la que todas las metaheurísticas se ejecutan paralelamente.

Se realizaron pruebas con 20 instancias (una por tamaño y tipo), distintas de las utilizadas en el aprendizaje. Cada instancia se resolvió 10 veces utilizando 100000 evaluaciones de la función objetivo, los periodos de intercambio están definidos por el intervalo [500, 600] y se tomó un α *corte* de 0.5. Los resultados se muestran en la Figura 6.17 y en la tabla 6.5. En ellos se puede observar un comportamiento muy similar con ambas arquitecturas, sin muchas diferencias significativas, aunque en algunos casos puntuales es la arquitectura monocapa la que obtiene mejores resultados y por tanto la que se utilizará en el resto del capítulo. Sin embargo, a pesar de habernos decantado por la arquitectura monocapa, los resultados obtenidos por la arquitectura bicapa son también interesantes y susceptibles de ser estudiados más en profundidad, y en ningún caso deben ser descartados.

Tabla 6.5: Resultados según la arquitectura

Tamaño	Tipo	KEPS^1	KEPS^2
500	unc span	$4{,}48_{0{,}81}$	$4{,}31_{0{,}81}$
	wea span	$1{,}57_{0{,}14}$	$1{,}60_{0{,}14}$
	str span	$1{,}01_{0{,}06}$	$0{,}99_{0{,}06}$
	pceil	$0{,}10_{0{,}00}$	$0{,}11_{0{,}00}$
	circle	$7{,}58_{0{,}46}$	$7{,}46_{0{,}46}$
1000	unc span	$8{,}46_{0{,}69}$	$9{,}69_{0{,}69}$
	wea span	$2{,}22_{0{,}21}$	$2{,}26_{0{,}21}$
	str span	$1{,}11_{0{,}04}$	$1{,}12_{0{,}04}$
	pceil	$0{,}11_{0{,}00}$	$0{,}11_{0{,}00}$
	circle	$9{,}73_{0{,}47}$	$9{,}91_{0{,}47}$
1500	unc span	$10{,}68_{1{,}03}$	$12{,}45_{1{,}03}$
	wea span	$2{,}84_{0{,}12}$	$2{,}78_{0{,}12}$
	str span	$1{,}23_{0{,}02}$	$1{,}21_{0{,}02}$
	pceil	$0{,}12_{0{,}00}$	$0{,}12_{0{,}00}$
	circle	$10{,}36_{0{,}31}$	$10{,}62_{0{,}31}$
2000	unc span	$11{,}18_{0{,}80}$	$12{,}76_{0{,}80}$
	wea span	$3{,}11_{0{,}18}$	$3{,}02_{0{,}18}$
	str span	$1{,}26_{0{,}02}$	$1{,}24_{0{,}02}$
	pceil	$0{,}12_{0{,}00}$	$0{,}12_{0{,}00}$
	circle	$10{,}15_{0{,}31}$	$10{,}38_{0{,}31}$

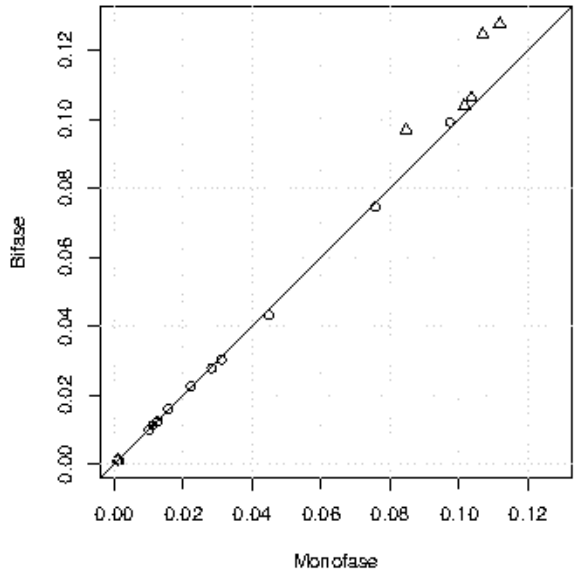


Figura 6.17: Comparativa entre KEPS^2 y KEPS^1.

6.5. Análisis de la influencia del esquema de cooperación

En esta sección se comparan los resultados obtenidos utilizando los dos esquemas de cooperación propuestos, el basado en memoria o KEPS^1_{Mem}, que utiliza una selección de parámetros ad-hoc y una regla de intercambio de soluciones basada en una memoria de la ejecución actual, y el basado en conocimiento previo o $\text{KEPS}^1_{CP\ T}$, que se sirve del conocimiento extraído de ejecuciones previas para seleccionar los parámetros más convenientes para cada metaheurística y controlar el intercambio de soluciones.

Se realizaron pruebas con 20 instancias (una por tamaño y tipo), distintas de las utilizadas en el aprendizaje. Cada instancia se resolvió 10 veces utilizando 100000 evaluaciones de la función objetivo, los periodos de intercambio están definidos por el intervalo $[500, 600]$ y se tomó un α *corte* de 0.5. Los resultados se muestran en la tabla 6.6 y en la figura 6.18. Como puede observarse la mayoría de puntos se encuentran por debajo de la diagonal representados por triángulos lo que indica que el esquema basado en conocimiento obtiene mejores resultados que el basado en memoria, y que la información extraída de las ejecuciones anteriores supone un conocimiento útil para la resolución del problema.

Tabla 6.6: Resultados según el esquema de cooperación

Tamaño	Tipo	$KEPS^1_{CP-T}$	$KEPS^1_{Mem}$
500	unc span	$3,84_{0,60}$	$6,24_{0,65}$
	wea span	$1,60_{0,34}$	$3,93_{0,23}$
	str span	$0,99_{0,07}$	$1,02_{0,06}$
	pceil	$0,11_{0,00}$	$0,10_{0,01}$
	circle	$7,27_{0,48}$	$7,42_{0,50}$
1000	unc span	$8,70_{1,24}$	$12,80_{1,57}$
	wea span	$2,23_{0,20}$	$4,16_{0,28}$
	str span	$1,12_{0,05}$	$1,19_{0,04}$
	pceil	$0,11_{0,00}$	$0,11_{0,00}$
	circle	$9,49_{0,30}$	$9,90_{0,59}$
1500	unc span	$10,86_{0,67}$	$18,96_{1,26}$
	wea span	$2,74_{0,12}$	$4,64_{0,12}$
	str span	$1,22_{0,03}$	$1,30_{0,04}$
	pceil	$0,12_{0,00}$	$0,12_{0,00}$
	circle	$10,14_{0,41}$	$11,90_{0,52}$
2000	unc span	$11,33_{1,12}$	$22,20_{0,91}$
	wea span	$3,14_{0,17}$	$4,66_{0,14}$
	str span	$1,27_{0,02}$	$1,34_{0,03}$
	pceil	$0,12_{0,00}$	$0,13_{0,00}$
	circle	$10,10_{0,35}$	$13,34_{0,59}$

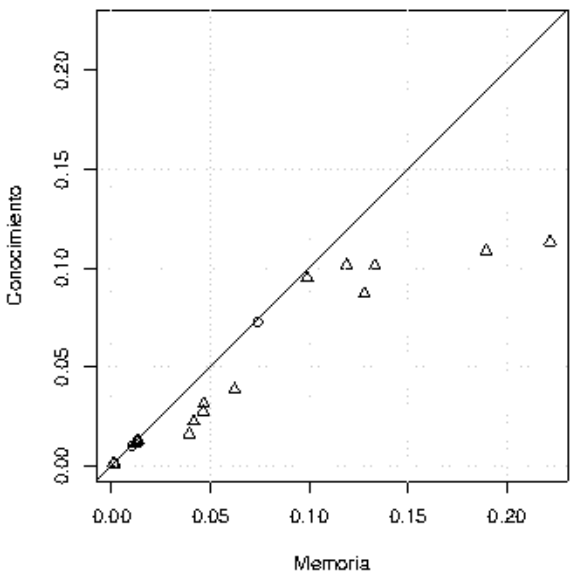


Figura 6.18: Comparativa entre los esquemas de cooperación $KEPS^1_{Mem}$ y $KEPS^1_{CP\ \ T}$

6.6. Estudio del impacto de la frecuencia de intercambios y del α-corte

En esta sección se estudia de que manera afecta al sistema propuesto la elección de la frecuencia de intercambio y el α *corte*. Para ello se realizaron pruebas con 20 instancias (una por tamaño y tipo), distintas de las utilizadas en el aprendizaje. Cada instancia se resolvió 10 veces utilizando 100000 evaluaciones de la función objetivo, variando la frecuencia de intercambio y el α *corte*. Los resultados se muestran en la tabla 6.7. En cada celda se muestra el error medio obtenido para todas las instancias y como subíndice la desviación típica. En esta tabla se pueden observar dos tendencias claras. Por un lado el α *corte* idóneo en todos los casos es 0.1, si se reduce o se aumenta, los resultados empeoran. Por otro lado, en cuanto a frecuencias, una frecuencia muy alta mejora los resultados, probablemente debido a la gran compartición de información que favorece la localización de zonas prometedoras con velocidad, si la frecuencia se reduce esta capacidad se pierde empeorando los resultados, que ya no mejorarán hasta que se reduzca tanto como para dejar a las metaheurísticas converger por sí mismas. En este caso los intercambios de información contienen una información mucho más depurada, lo que explicaría la mejora en los resultados. Por último cabe resaltar los resultados obtenidos cuando no se permiten intercambios, es decir una metaheurística paralela pero no cooperativa que se muestran en la última fila de la tabla 6.7 y que son independiente del α-corte, puesto que no tiene sentido en este caso. En este caso se puede observar cómo los resultados obtenidos por la estrategia cooperativa siempre son mejores independientemente de la configuración de frecuencia y α *corte*.

Tabla 6.7: Resultados según frecuencia de intercambios y α-corte

Frecuencia	0.01	0.1	0.2	0.5	0.9
100-150	$4.41_{0.23}$	$4.37_{0.24}$	$4.50_{0.27}$	$4.88_{0.25}$	$4.98_{0.23}$
500-600	$4.46_{0.30}$	$4.50_{0.29}$	$4.50_{0.29}$	$4.66_{0.29}$	$4.82_{0.27}$
1500-1700	$4.54_{0.26}$	$4.55_{0.29}$	$4.66_{0.31}$	$4.78_{0.29}$	$4.74_{0.25}$
5000-5500	$4.20_{0.29}$	$4.19_{0.27}$	$4.24_{0.28}$	$4.26_{0.26}$	$4.26_{0.26}$
Sin intercambios			$5,00_{0,29}$		

6.7. Conclusiones

Una vez que hemos definido en el Capítulo 5 el marco general para la construcción de metaheurísticas cooperativas con conocimiento previo, en este capítulo hemos profundizado en el estudio del impacto de los distintos parámetros que influyen en su diseño y, por tanto,

en su comportamiento.

Se ha estudiado la influencia de la composición de la metaheurística cooperativa, comparando diversas composiciones homogéneas con una heterogénea y distintas metaheurísticas individuales. En estas comparaciones, se concluye cómo una metaheurística cooperativa homogénea siempre supera en rendimiento a su homóloga individual, y cómo la metaheurística heterogénea es la que mejores resultados obtiene.

Se ha estudiado qué tipo de arquitectura resulta más adecuada si 1) una arquitectura bicapa, con dos fases secuenciales, una relativa a metaheurísticas basadas en poblaciones y otra relativa a metaheurísticas basadas en trayectorias, o 2) una arquitectura monocapa, en la que todas las metaheurísticas se ejecutan en paralelo. Los resultados demuestran que ambas arquitecturas son válidas, obteniendo resultados similares.

Se ha comprobado qué esquema de cooperación produce mejores resultados, si uno basado en la memoria de la ejecución o en conocimento adquirido previamente. Los resultados muestran una mejora sensible al utilizar la segunda aproximación, por lo que se concluye que la utilización de un proceso de aprendizaje previo ayuda a las metaheurísticas cooperativas.

Por último, se ha comprobado la influencia del número de intercambios realizados, observándose que un número demasiado alto puede llevar a malos resultados, mientras que pocos intercambios también influyen negativamente. Un término intermedio de intercambios obtiene resultados muy satisfactorios.

Capítulo 7

La metaheurística KEPS con mecanismo de cooperación controlado por tiempo

Las metaheurísticas paralelas, y entre ellas las cooperativas, surgieron inicialmente como una manera de reducir el tiempo empleado en la ejecución de las metaheurísticas de tal manera que se pudieran obtener mejores soluciones utilizando el mínimo tiempo posible. Puesto que la estrategia propuesta se engloba dentro de las metaheurísticas cooperativas y por tanto de las paralelas, se ha considerado importante realizar unas pruebas que tengan en cuenta el tiempo y cuyo mecanismo de cooperación esté controlado por tiempo en lugar de por evaluaciones de la función objetivo. En este capítulo nos proponemos demostrar la validez de la estrategia propuesta en un entorno en el que el tiempo de ejecución sea un factor importante. Para ello se utilizará como problema test el problema de la mochila-{0,1}, y se realizarán diversas pruebas comparando distintas metaheurísticas aisladas y otras estrategias cooperativas más simples con la estrategia propuesta $\text{KEPS}^1_{C\,P\;\;T}$.

7.1. Un problema de test

El problema que se utilizará para realizar las pruebas concernientes al tiempo es el ya utilizado problema de la mochila-{0,1}. Este problema se define formalmente como: Dado un conjunto de ítems, cada una con un coste y un beneficio asociados, determinar un subconjunto tal que el coste total sea menor que un límite dado y que el beneficio total sea tan grande como sea posible (más información en capítulo 6). Su formalización matemática es:

$$\begin{array}{ll} Max & \sum_{i=1}^{n} p_i \times x_i \\ s.a & \sum_{i=1}^{n} w_i \times x_i \;\leq\; C \\ & x_i \in \{0,1\}\; i = 1,\ldots,n \end{array}$$

donde

- n es el número de ítems.
- x_i indica si un ítem i está incluido en la mochila o no.
- p_i es el beneficio asociado al ítem i.
- $w_i \in [0, \ldots, r]$ es el peso del ítem i.
- C es la capacidad máxima de la mochila.

Se asume que $w_i < C, \ \forall i$ y $\sum_{i=1}^{n} w_i > C$.

7.2. Construcción de la metaheurística cooperativa

En esta sección se explican aspectos importantes en la construcción de la metaheurística cooperativa, como son las metaheurísticas que formarán parte de ella, y el proceso de extracción de conocimiento seguido.

7.2.1. Metaheurísticas integradas

Para dar forma a la metaheurística cooperativa debemos elegir las metaheurísticas que la conformarán. En este caso se han seleccionado tres metaheurísticas, dos basadas en trayectorias, búsqueda tabú y temple simulado, y una basada en poblaciones, algoritmo genético. Las tres metaheurísticas siguen la misma implementación mostrada en el capítulo 6.

7.2.2. El proceso de extracción del conocimiento

Para obtener la configuración de $\text{KEPS}^{1}_{CP\ \ T}$ es necesario aplicar el proceso de extracción inteligente del conocimiento. Como se explicó previamente, en la fase preparación de datos, se deben resolver distintas instancias para obtener información que pueda ayudar al coordinador a adaptarse a las posibles instancias. En el caso que atañe a este capítulo se utilizaron 1000 instancias, generadas con $R = 10^3$ y $R = 10^4$, cuatro tamaños de problema, $n = (500, 1000, 1500, 2000)$, los 5 tipos de instancias definidos anteriormente y h generado aleatoriamente entre $[1, 100]$. Además, las metaheurísticas empleadas usaron los siguientes valores de parámetros:

- Algoritmo Genético:
  - Operador de selección: Ruleta.
  - Probabilidad de cruce: 0.6 y 0.4.

- Probabilidad de mutación: 0.1 y 0.01.
- Número de individuos: 200 y 300.

- Búsqueda Tabú:
  - Tamaño de la lista tabú: 10, 100, 1000.
  - Umbral de diversificación: 1, 10 y 50..

- Temple Simulado:
  - Función de enfriamiento: Tangente hiperbólica, coseno hiperbólico, raíz cuadrada y exponencial.
  - Número de iteraciones internas: 9000 y 10000.
  - Número de soluciones generadas antes de la selección: 1.

Tras reunir la información y obtener las bases de datos crudas refinándolas posteriormente para obtener las bases de datos refinadas, se obtuvieron los árboles que configuran al agente coordinador.

7.3. Resultados Computacionales

En esta sección se realizan distintos tests para probar la validez de la aproximación. Inicialmente se comprobará el rendimiento de las metaheurísticas aisladas comparándolas con un sistema paralelo no cooperativo, consistente en ejecutar todas las metaheurísticas en paralelo y seleccionar la mejor solución obtenida. A continuación se probará la eficiencia/eficacia de $\text{KEPS}^1_{\mathcal{CP}\ \mathcal{T}}$ y la utilidad de la incorporación de conocimiento, cuando el control de cooperación se realiza con una variable tiempo.

Para realizar estos tests, inicialmente se utilizan instancias con $R = 10^3$ para comparar los desempeños de las distintas metaheurísticas aisladas y los sistemas no cooperativos y cooperativos. Después, se utilizan instancias con $R = 10^4$ para confirmar los resultados obtenidos. En las siguientes subsecciones se muestran la configuración de los tests y los resultados obtenidos.

En cada subsección y una vez que se hayan mostrado los resultados, se realiza un análisis de los métodos usando técnicas estadísticas. Siguiendo la metodología propuesta en [123], que utiliza tests no paramétricos. En particular se utiliza el test de Wilcoxon para comparar dos métodos. Este test es un procedimiento no paramétrico para realizar comparaciones pareadas entre dos algoritmos. Es el análogo no paramétrico del t-test pareado. Por tanto es un test paread que trata de detectar diferencias significativas entre dos aproximaciones.

Cuando se comparan varios métodos (más de dos), se utiliza el test de Friedman y el método Benjamin-Hochberger como test post-hoc. Este último método es más potente que el test Bonferroni-Dunn, el test Holm y el método Hochberger. El test de Friedman es el equivalente no paramétrico al ANOVA. Como hipótesis nula, considera que todos los algoritmos son equivalentes, de tal manera que el rechazo de esta hipótesis implica la existencia de diferencias en el rendimiento de los algoritmos estudiados. Después de esto se puede usar el test post-hoc para discernir si el algoritmo propuesto presenta diferencias estadísticas con respecto al resto de métodos de la comparación.

7.3.1. Configuración de los experimentos

Para llevar a cabo los tests se utilizó un conjunto de instancias compuesto de 100 instancias test (distintas de las de entrenamiento). Entre ellas hay 20 instancias por cada combinación tamaño $(500, 1000, 1500, 2000)$ y tipo (unc span, wea span, str span, pceil, circle).

Todos los tests se ejecutaron en un Intel core2 Quad 1.66GHz con 2GB de memoria. Cada instancia se resolvión 10 veces durante 90 segundos, y los resultados muestran la media de error ($error = 100 \times \frac{optimo\ \ valorobtenido}{optimo}$), y como subíndice la desviación típica.

Adicionalmente, se utilizarán los siguientes parámetros del sistema:

- Arquitectura monofase.
- Esquema de cooperación basado en conocimiento.
- $suficiente$ es un conjunto fuzzy con función de pertenencia trapezoidal, donde $(a, b, c, d) = (0.005, 0.01, 1, 1)$.
- α $\ corte$ toma el valor 0,01.
- Frecuencia de intercambio cada 250 milisegundos.

Destacamos que al controlar la frecuencia de intercambios con una variable tiempo provoca que las metaheurísticas, al ser paradas todas al mismo tiempo, convergan a las soluciones de manera muy desigual (cada metaheurística habrá realizado un número de evaluaciones de la función fitness distinto a las demás).

7.3.2. Rendimiento de las metaheurísticas

En esta subsección se muestran los resultados obtenidos por las metaheurísticas aisladas. Se decidió que cada metaheurística se ejecutara con los mejores parámetros globales obtenidos durante el proceso de extracción del conocimiento. La Tabla 7.1 muestra los resultados obtenidos por las metaheurísticas individuales discriminadas por tamaño y tipo de instancia.

También se muestran los resultados de sistemas no cooperativos teóricos, en los cuales se ejecutan las distintas metaheurísticas en paralelo tomando como salida la mejor solución obtenida. El primer sistema (NCS-NI) no utiliza ningún tipo de información del proceso de extracción del conocimiento, el segundo (NCS-BP) usa los mejores parámetros globales obtenidos durante este.

Tabla 7.1: Metaheurísticas aisladas y sistemas no cooperativos sobre instancias $R = 10^3$

Tipo	Tamaño	Algoritmo Genético	Búsqueda Tabú	Temple Simulado	NCS-NI	NCS-BP
Unc Span	500	$5.07_{0.99}$	$1.21_{0.55}$	$0.21_{0.15}$	$0.70_{0.31}$	$0.17_{0.12}$
	1000	$18.06_{0.89}$	$0.90_{0.24}$	$0.21_{0.08}$	$0.71_{0.37}$	$0.14_{0.06}$
	1500	$25.14_{0.80}$	$1.17_{0.21}$	$0.20_{0.07}$	$0.98_{0.37}$	$0.16_{0.04}$
	2000	$28.66_{0.92}$	$1.05_{0.30}$	$0.13_{0.04}$	$1.01_{0.41}$	$0.11_{0.03}$
Wea Span	500	$1.87_{0.30}$	$3.10_{0.84}$	$2.02_{0.35}$	$1.51_{0.28}$	$1.67_{0.28}$
	1000	$6.17_{0.35}$	$1.81_{0.32}$	$2.18_{0.27}$	$1.59_{0.30}$	$1.44_{0.29}$
	1500	$9.48_{0.72}$	$2.35_{0.37}$	$2.31_{0.32}$	$1.95_{0.30}$	$1.71_{0.25}$
	2000	$12.03_{0.46}$	$2.62_{0.40}$	$2.60_{0.21}$	$2.06_{0.23}$	$1.95_{0.25}$
Str Span	500	$3.00_{0.29}$	$8.58_{1.97}$	$2.54_{0.24}$	$4.05_{0.82}$	$2.51_{0.25}$
	1000	$7.86_{0.34}$	$9.61_{1.29}$	$2.47_{0.16}$	$4.95_{1.63}$	$2.45_{0.16}$
	1500	$12.50_{0.97}$	$9.94_{0.58}$	$2.48_{0.15}$	$6.05_{2.06}$	$2.46_{0.15}$
	2000	$16.00_{0.89}$	$9.83_{1.08}$	$2.26_{0.08}$	$6.53_{2.27}$	$2.24_{0.08}$
Pceil	500	$1.15_{0.18}$	$0.47_{0.12}$	$0.30_{0.02}$	$0.41_{0.07}$	$0.29_{0.03}$
	1000	$4.96_{0.19}$	$0.58_{0.02}$	$0.36_{0.01}$	$0.43_{0.06}$	$0.36_{0.01}$
	1500	$9.88_{0.42}$	$0.64_{0.01}$	$0.39_{0.01}$	$0.45_{0.07}$	$0.39_{0.01}$
	2000	$10.75_{0.36}$	$0.63_{0.01}$	$0.38_{0.01}$	$0.46_{0.06}$	$0.38_{0.01}$
Circle	500	$7.12_{0.62}$	$19.59_{4.72}$	$7.00_{0.62}$	$14.34_{3.51}$	$7.00_{0.62}$
	1000	$19.86_{1.28}$	$24.94_{1.71}$	$7.82_{0.98}$	$15.33_{4.59}$	$7.82_{0.98}$
	1500	$27.92_{1.13}$	$25.52_{1.46}$	$8.22_{0.79}$	$17.41_{5.72}$	$8.22_{0.79}$
	2000	$32.84_{0.78}$	$25.39_{1.13}$	$7.97_{0.49}$	$14.92_{5.14}$	$7.97_{0.49}$

En los resultados se observa como la peor metaheurística en términos de rendimiento es el Algoritmo Genético, cuyas diferencias en error son estadísticamente mayores que las del resto de acuerdo con el test no paramétrico de Wilcoxon (se utiliza un nivel de confianza del 95 % en todas las aplicaciones de este test no paramétrico en este capítulo). Por otro lado la mejor metaheurística individual es, en general, el Temple Simulado, que obtiene mejor rendimiento que el resto en la mayoría de casos, siendo las diferencias significativas de acuerdo con el test de Wilcoxon.

Con respecto a los sistemas paralelos, NCS-NI, como era esperado, obtiene peores resultados que la mejor metaheurística individual (también estadísticamente) puesto que tiene menos información (no utiliza los mejores parámetros). En contraste, NCS-BP obtiene un mejor rendimiento que la mejor metaheurística, con diferencias significativas. Aunque en general, la mejor metaheurística es el temple simulado, NCS-BP obtiene mejores resultados porque para ciertas instancias es otra metaheurística la que obtiene los mejores resultados.

7.3.3. Comparando metaheurísticas paralelas

En esta sección se incorporan a la comparación algunas estrategias cooperativas. La primera es CS-WB, la cual no utiliza ningún tipo de conocimiento y utiliza un esquema de cooperación básico en el que en cada paso la metaheurística que obtiene la mejor solución la envía a aquella que obtenga la peor. La segunda es CS-FPL, que es similar al sistema propuesto pero en lugar de utilizar árboles para la selección de parámetros y pesos se utilizan valores globales obtenidos analizando las bases de datos de aprendizaje.

La Tabla 7.2 muestra los resultados obtenidos por los dos sistema cooperativos. También se comparan los resultados con los mejores obtenidos por los sistemas anteriores (Temple Simulado y NCS-BP). En la Tabla 7.2 se puede observar como los sistemas cooperativos obtienen mejor desempeño que las metaheurísticas individuales y los sistemas no cooperativos. Además las diferencias de error son estadísticamente significativas.

Tabla 7.2: Sistemas paralelos no cooperativos y cooperativos para instancias $R = 10^3$

Type	Size	Temple Simulado	NCS-BP	CS-WB	CS-FPL
Unc Span	500	0.21 0.15	0.17 0.12	0.01 0.01	0.04 0.02
	1000	0.21 0.06	0.14 0.06	0.16 0.11	0.22 0.09
	1500	0.20 0.07	0.16 0.04	0.39 0.32	0.32 0.24
	2000	0.13 0.04	0.11 0.03	0.49 0.42	0.33 0.29
Wea Span	500	2.02 0.35	1.67 0.26	0.66 0.33	0.56 0.11
	1000	2.18 0.27	1.44 0.29	0.85 0.31	0.84 0.11
	1500	2.31 0.32	1.71 0.25	1.05 0.43	1.11 0.14
	2000	2.60 0.21	1.95 0.25	1.13 0.46	1.26 0.13
Str Span	500	2.54 0.24	2.51 0.25	0.96 0.43	1.12 0.14
	1000	2.47 0.16	2.45 0.16	1.90 1.61	1.19 0.11
	1500	2.48 0.15	2.46 0.15	2.19 2.01	1.36 0.17
	2000	2.26 0.06	2.24 0.06	1.93 0.91	1.60 0.11
Pceil	500	0.30 0.02	0.29 0.03	0.21 0.06	0.19 0.03
	1000	0.36 0.01	0.36 0.01	0.22 0.04	0.24 0.01
	1500	0.39 0.01	0.39 0.01	0.26 0.05	0.27 0.02
	2000	0.38 0.01	0.38 0.01	0.29 0.07	0.27 0.01
Circle	500	7.00 0.82	7.00 0.82	5.77 4.15	3.48 0.40
	1000	7.82 0.96	7.82 0.96	7.90 4.06	4.99 0.39
	1500	8.22 0.79	8.22 0.79	7.37 2.11	5.96 0.41
	2000	7.97 0.49	7.97 0.49	9.38 4.36	6.72 0.34

También se puede observar que el sistema cooperativo que muestra el mejor comportamiento es el que utiliza conocimiento previo. Las diferencias de error entre los dos sistemas, CS-WB y CS-FPL, son significativas, siendo el error obtenido por CS-FPL menor que el obtenido por CS-WB. Como puede observarse CS-FPL obtiene mejores resultados que CS-WB en 13 instancias de 20, y con una mejor desviación típica. En aquellas instancias en las que las soluciones presentadas por CS-WB son mejores que las de CS-FPL, las diferencias en ratio

de error son bastante pequeñas, pero la desviación típica de CS-WB es mayor (excepto para las instancias Unc Span de tamaño 500) lo que indica que CS-WB no es robusto. Además, CS-FPL obtiene un comportamiento razonable en las instancias más difíciles. Basándonos en estas razones se puede concluir que el modelo seleccionado para CS-FPL es eficiente y robusto, y que la información incluida en su base de reglas es útil. Sin embargo, como puede verse en los resultados, para algunas instancias, CS-WB obtiene mejores resultados. Por tanto está justificada la inclusión de nueva información a las reglas, de tal manera que pueda adaptarse mejor a las instancias, y por tanto obtener un mejor comportamiento global. Lo cual se obtiene con los árboles obtenidos del proceso de extracción del conocimiento.

7.3.4. Rendimiento de KEPS

En esta sección se estudian los beneficios del comportamiento adaptativo que surge de la utilización de los árboles para seleccionar los parámetros del sistema y la importancia de las metaheurísticas, que ayudan a mejorar su adaptabilidad y robustez puesto que tienen en cuenta la instancia que se está resolviendo. Los resultados de esta comparación se muestran en la Tabla 7.3, donde se puede apreciar como $\text{KEPS}^1_{CP\ T}$ mejora a CS-WB y CS-FPL en casi todos los casos cuando resolvemos las instancias de tipo R^3 y R^4.

Tabla 7.3: Incorporación de conocimiento al sistema

		$R = 10^3$			$R = 10^4$		
Type	Size	CS-WB	CS-FPL	KEPS	CS-WB	CS-FPL	KEPS^1_T
Unc Span	500	0.01 0.01	0.04 0.02	0.01 0.03	0.16 0.24	0.00 0.00	0.00 0.00
	1000	0.16 0.11	0.22 0.09	0.14 0.05	0.67 0.89	0.00 0.00	0.00 0.00
	1500	0.39 0.32	0.32 0.24	0.23 0.07	1.18 1.24	0.02 0.03	0.01 0.02
	2000	0.49 0.42	0.33 0.29	0.13 0.07	2.13 1.78	0.18 0.02	0.02 0.01
Wea Span	500	0.66 0.33	0.56 0.11	0.52 0.09	0.26 0.21	0.11 0.05	0.11 0.05
	1000	0.85 0.31	0.84 0.11	0.75 0.12	0.65 0.36	0.27 0.12	0.27 0.12
	1500	1.05 0.43	1.11 0.14	1.00 0.13	1.12 0.46	0.34 0.06	0.34 0.06
	2000	1.13 0.48	1.26 0.13	1.15 0.10	1.28 0.41	0.84 0.13	0.31 0.07
Str Span	500	0.96 0.43	1.12 0.14	0.78 0.14	2.86 1.99	0.30 0.30	0.25 0.21
	1000	1.90 1.61	1.19 0.11	1.13 0.09	3.62 2.48	2.30 0.41	1.03 0.28
	1500	2.19 2.01	1.36 0.17	1.22 0.12	5.62 2.53	2.08 0.42	1.45 0.29
	2000	1.93 0.91	1.60 0.11	1.48 0.07	6.55 2.91	4.85 0.59	2.00 0.05
Pcell	500	0.21 0.06	0.19 0.03	0.16 0.03	0.04 0.01	0.04 0.00	0.04 0.00
	1000	0.22 0.04	0.24 0.01	0.22 0.02	0.05 0.01	0.05 0.00	0.05 0.00
	1500	0.26 0.05	0.27 0.02	0.25 0.02	0.06 0.01	0.08 0.00	0.06 0.00
	2000	0.29 0.07	0.27 0.01	0.26 0.02	0.06 0.01	0.07 0.00	0.06 0.00
Circle	500	5.77 4.15	3.48 0.40	3.09 0.42	7.22 3.15	1.16 0.32	1.13 0.32
	1000	7.90 4.08	4.99 0.39	4.52 0.34	12.27 4.57	3.61 0.40	2.80 0.37
	1500	7.37 2.11	5.96 0.41	5.55 0.37	7.48 2.15	9.23 0.76	5.14 0.47
	2000	9.38 4.36	6.72 0.34	6.31 0.29	18.21 5.48	16.80 0.79	9.03 0.63

Como se puede observar, $\text{KEPS}^1_{CP\ T}$ tiene mejor comportamiento que CS-FPL debido a su adaptabilidad. CS-FPL siempre tiene la misma configuración para todas las instancias, mientra que $\text{KEPS}^1_{CP\ T}$ utiliza un conjunto de árboles fuzzy para adaptar sus parámetros a las distintas instancias. Además, de acuerdo con el test de Wilcoxon, las diferencias son significativas. Como resultado adicional en la Tabla 7.4 se muestra para cada sistema el porcentaje de instancias que resolvió con un error menor que 1%. Todos estos resultados sirven para mostrar la utilidad de la información incorporada para obtener la configuración del sistema, y para demostrar que $\text{KEPS}^1_{CP\ T}$ es estable y obtiene un comportamiento satisfactorio.

Tabla 7.4: Instancias resueltas con error $< 1\%$

$R = 10^3$			$R = 10^4$		
CS-WB	**CS-FPL**	$\textbf{KEPS}^1_{CP-T}$	**CS-WB**	**CS-FPL**	$\textbf{KEPS}^1_{CP-T}$
59	62	67	57	71	75

7.4. Conclusiones

En este capítulo se ha podido observar que la cooperación y el uso del conocimiento adquirido por el coordinador para controlar la ejecución de las metaheurísticas individuales son muy beneficiosos cuando el tiempo es una de las variables que gobiernan la cooperación.

En concreto, cuando controlamos la frecuencia de intercambio para la cooperación mediante una variable tiempo, la metaheurística cooperativa $\text{KEPS}^1_{CP\ T}$ es muy estable y obtiene resultados muy satisfactorios. La cooperación en función del tiempo produce una mejora sustancial en los resultados de $\text{KEPS}^1_{CP\ T}$ comparándolos con los obtenidos por las metaheurísticas individuales. Estos resultados nos muestran que la cooperación es muy importante y eficiente provocando que, aunque las metaheurísticas individuales no tuvieran una buena convergencia a buenas soluciones, la metaheurística cooperativa obtenga un buen rendimiento y buenas soluciones.

En el Capítulo 6 se concluyó, y aquí reforzamos, que KEPS en sus versiones, es una metaheurística eficiente y eficaz.

Parte III

APLICACIONES DE LAS METAHEURISTICAS COOPERATIVAS

Capítulo 8

Un problema de e-Learning

En las últimas décadas, tanto las instituciones académicas como la industria, han hecho uso de distintos métodos de aprendizaje para mejorar las capacidades y conocimientos de alumnos/trabajadores, de tal manera que su rendimiento y competitividad mejorara en el nuevo contexto económico. Sin embargo, la significativa velocidad a la que se genera nuevo conocimiento y su volatilidad, debido al crecimiento exponencial de las fuentes de información, ha complicado la actividad de los alumnos. Al mismo tiempo, las nuevas tecnologías ofrecen, si se usan de un modo adecuado, muchas posibilidades para el diseño eficiente de los escenarios de aprendizaje. Por todas estas razones surge la necesidad de obtener soluciones de aprendizaje capaces de generar experiencias de aprendizaje eficientes, personalizadas y flexibles. Desde este punto de vista, es necesario el empleo de herramientas capaces de analizar las necesidades y preferencias del alumno y, en consecuencia, personalizar su adquisición de nuevas habilidades y conocimientos. En este capítulo se plantea la resolución de este problema utilizando una representación ontológica del entorno de aprendizaje y la estrategia propuesta, concretamente usando la arquitectura bicapa y el esquema de cooperación basado en conocimiento, o lo que es lo mismo, $\text{KEPS}^2_{\mathcal{CP}\ \ \mathcal{T}}$.

8.1. El problema de la presentación de aprendizaje

8.1.1. Introducción

Durante muchos años, las mejoras tecnológicas aplicadas al aprendizaje humano [234] se han realizado siguiendo el paradigma de transmisión de información, es decir, un profesor proporciona contenidos educacionales que deben ser transferidos a un grupo de alumnos, y los alumnos consumen estos contenidos de un modo pasivo. Consecuentemente, la mayoría de soluciones de e-learning simplemente "digitalizan" esta aproximación, lo que resulta en

plataformas software de aprendizaje a distancia. Estas plataformas computan *presentaciones de aprendizaje* teniendo en cuenta un solo tipo de input, los "recursos educacionales". Esta limitación, junto con otras como el soporte para pedagogías, la contextualización de la experiencia de aprendizaje o el compromiso del alumno en las actividades, se consideran las principales barreras para la adopción del e-Learning y trazar el camino para futuros desarrollos.

En particular, las soluciones actuales de e-learning muestran los siguientes problemas:

1. Fuerzan a los alumnos a aprender sin tener en cuenta sus disposiciones o preferencias;
2. Obligan a los alumnos a aprender y a los profesores a enseñar usando una aproximación predefinida;
3. No soportan el impulso proporcionado por la Web 2.0.

Para afrontar estas debilidades, las soluciones de e-Learning modernas deben representar una evolución real donde los mayores esfuerzos se deben centrar en dar soporte a todo proceso de aprendizaje, no solo partes especificas del mismo (es decir la administración de contenidos, la entrega de contenidos, etc.) y deben incorporar herramientas como Blogs (usados para compartir ideas), Wikis (usados como mecanismo para construir conocimiento de un modo colaborativo), Podcast (usados para distribuir contenido multimedia en Internet) y otras aplicaciones de compartición web (ej: Flickr, YouTube, del.icio.us, etc.) que permitan a los usuarios de e-Learning trabajar, enseñar, aprender, hacer negocios, etc. Los procesos de e-learning que satisfacen estos requerimientos, utilizando las herramientas anteriormente descritas coherentemente, pertenecen a los escenarios de e-learning 2.0 [110]. Consecuentemente, la siguiente generación de sistemas de administración de aprendizaje (SAA) [92] deben soportar las nuevas tendencias en e-Learning usando modelos apropiados y procesos que creen experiencias de e-Learning (una secuencia estructurada de actividades de aprendizaje compuestas de contenidos y servicios capaces de facilitar a los alumnos la adquisición de una serie de competencias acerca de un dominio específico) adaptadas a las expectativas del alumno de un modo inteligente y dinámico.

Por tanto, se deben definir experiencias de e-Learning personalizadas capaces de maximizar el nivel de comprensión de los alumnos con respecto a objetivos de aprendizaje específicos. Esto requiere un modelo de representación para describir tanto los dominios educacionales como las características de los alumnos. En este escenario, las metodologías ontológicas [136] modelan y representan los contextos de e-learning realizando una conceptualización de un dominio educacional por medio de la identificación de sus asignaturas relevantes y su organización usando un conjunto fijo de relaciones [98]. El análisis de relaciones ontológicas permite

una forma conveniente de enlazar asignaturas con actividades de aprendizaje para definir experiencias de e-learning personalizadas. Este problema de enlace se ha llamado *problema de presentación de aprendizaje* (PPA). Una posible aproximación para afrontar PPA es considerarlo como una versión del problema de la localización de facilidades no-capacitado (PLFN) [79] y resolverlo a través de un algoritmo determinista específico. Esta aproximación es conveniente cuando se considera un repositorio con un número limitado de actividades de aprendizaje y caminos de aprendizaje con pocas asignaturas. Pero cuando el ambiente de ejecución es la Web 2.0 [200] con repositorios distribuidos y los caminos de aprendizaje compuestos por muchas asignaturas se necesita investigar otras soluciones. En particular, una aproximación metaheurística, como $\text{KEPS}^2_{C,P,T}$, puede proporcionar resultados cercanos al óptimo utilizando recursos muy reducidos.

8.1.2. El entorno de e-Learning ontológico

El sistema de e-Learning propuesto se basa en un conjunto de modelos capaces de representar las principales entidades envueltas en el proceso de enseñanza/aprendizaje y en un conjunto de metodologías que se apoyan en ellos para generar experiencias de aprendizaje individualizadas con respecto a los objetivos de aprendizaje, el conocimiento anterior del alumno y sus preferencias de aprendizaje. Los tres modelos adoptados por nuestra solución, basados en el *Modelo de Aprendizaje* [10], son:

- El *modelo de dominio*. Este modelo representa el dominio de conocimiento que se pretende enseñar usando conceptos, relaciones entre conceptos y preferencias de enseñanza conectadas con conceptos.
- El *modelo de actividad de aprendizaje*. Este modelo representa un único objeto o servicio de aprendizaje que puede ser usado como bloque de construcción para generar experiencias de aprendizaje [168].
- El *modelo de unidad de aprendizaje*. Este modelo representa un experiencia de aprendizaje personalizada completa para un solo alumno y compuesta por un conjunto de conceptos objetivo y una secuencia de actividades de aprendizaje necesarias para aprender los conceptos objetivo.

El sistema utiliza estos modelos para automatizar algunas de las fases del proceso de enseñanza/aprendizaje. En particular, el profesor puede inicializar una unidad de aprendizaje fijando conceptos objetivo y asociando uno o más alumnos a este (en aprendizaje auto-dirigido, los alumnos hacen esto ellos mismos). Entonces, el sistema genera un *camino de aprendizaje*

(secuencia de conceptos que se quieren enseñar) para cada alumno usando un *algoritmo de generación de camino de aprendizaje*.

Una vez el camino de aprendizaje esté disponible, el sistema selecciona un fragmento del camino de aprendizaje y genera eficientemente la mejor *presentación de aprendizaje* (secuencia de actividades de aprendizaje) para cada alumno resolviendo el PPA. Entonces, el alumno lleva a cabo las actividades de aprendizaje y test de la presentación de aprendizaje hasta que termine. Cuando el alumno termina un fragmento de presentación, su modelo del alumno se actualiza en base a los resultados de las actividades de test y se genera un nuevo fragmento de la presentación de aprendizaje (posiblemente incluyendo actividades de recuperación para los conceptos que aun no comprenda).

8.1.2.1. El modelo de dominio

El modelo de dominio describe, en un modo procesable por el ordenador, la parte del dominio educacional que es relevante para la experiencia de e-Learning que se quiere definir, concretar y transmitir. El mecanismo usado se conoce como ontología, es decir, un esquema conceptual constituido por un vocabulario específico.

En esta aproximación, la ontología esta compuesta por un conjunto de conceptos (que representan los temas a enseñar) y un conjunto de relaciones entre conceptos (representado conexiones entre temas). Esta estructura se puede representar formalmente como una $(n+1)$-tupla $G(C, R_1, \ldots, R_n)$ donde C es el conjunto de nodos que representa los conceptos del dominio y cada R_i es un conjunto de arcos correspondiente con el i ésimo tipo de relación. Como un ejemplo se puede considerar un grafo de concepto $G(C, BT, IRB, SO)$ con relaciones de árbol BT, IRB y SO cuyo significado se explica a continuación (donde a y b son dos conceptos de C):

- $BT(a, b)$ significa que un concepto a pertenece a b, es decir, b se comprende si cada a tal que a pertenece a b se comprende (relación jerárquica).
- $IRB(a, b)$ significa que el concepto a es requerido por b, es decir, una condición necesaria para estudiar b es haber comprendido a (relación de orden).
- $SO(a, b)$ significa que el *orden sugerido* entre a y b es tal que a precede a b, es decir, para favorecer el aprendizaje, es deseable estudiar a antes que b.

La Figura 8.1 muestra un ejemplo de modelo de dominio en la didáctica de inteligencia artificial explotando las relaciones definidas anteriormente.

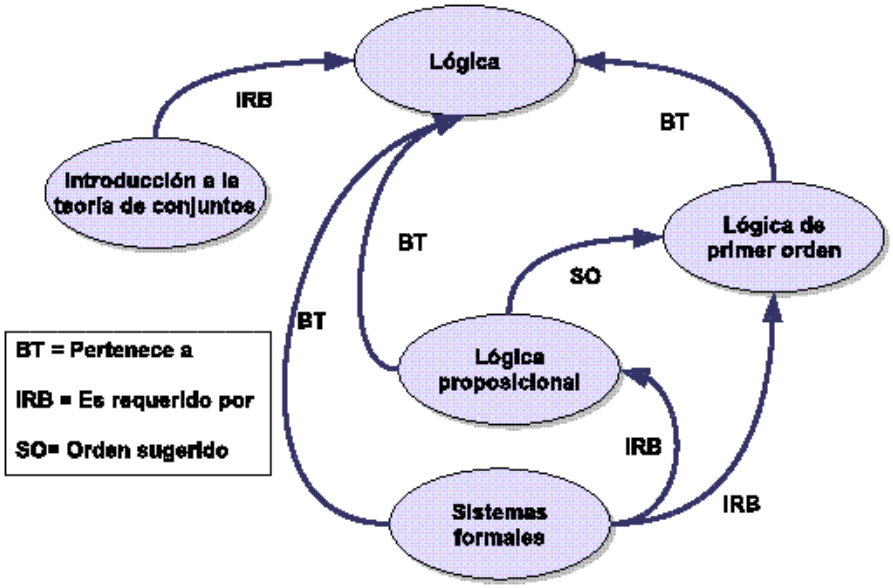


Figura 8.1: Un ejemplo de modelo de dominio.

Se puede añadir al modelo de dominio un conjunto de *preferencias de aprendizaje* para definir estrategias de enseñanza factibles que pueden ser aplicadas para cada concepto disponible. Estas preferencias se representan como una función $TP(C \times Props \times PropVals) \rightarrow [0, 10]$ donde $Props$ es el conjunto de propiedades didácticas y $PropVals$ es un conjunto de valores factibles para esas propiedades. La Tabla 8.1 proporciona un ejemplo (no exhaustivo) de propiedades didácticas y sus valores asociados. Se debe notar que TP se define solo para parejas de $Props$ y $PropVals$ pertenecientes a la misma fila en la Tabla 8.1.

Tabla 8.1: Ejemplo de propiedades didácticas y valores posibles.

Propiedades	Valores posibles
método didáctico	deductivo, inductivo, etc.
tipo de actividad	lectura de textos, videos, simulación discusión con compañeros, discusión con el profesor, etc.
nivel de interacción	alto, medio, bajo

8.1.2.2. El modelo del alumno

El alumno es el principal actor del proceso de aprendizaje completo y es representado con un estado cognitivo y un conjunto de preferencias de aprendizaje [167]. El *estado cognitivo* representa el conocimiento alcanzado por un alumno en un momento dado y se representa por una función $CS(C) \rightarrow [0,10]$ donde C es el conjunto de conceptos de un modelo de dominio dado. Dado un concepto c, $CS(c)$ indica el grado de conocimiento alcanzado por un alumno dado para ese concepto c. El valor 0 indica *sin conocimiento*, mientras que el valor 10 indica *conocimiento completo*. Si tal grado es mayor que un umbral dado θ entonces c se considera conocido, en otro caso se considera desconocido.

Las *preferencias de aprendizaje* proporcionan una evaluación de las estrategias de aprendizaje que se pueden adoptar para un determinado alumno. Se representan como una aplicación $LP(Props \times PropVals) \rightarrow [0,10]$ donde $Props$ y $PropVals$ son los mismos conjuntos definidos en la Tabla 8.1 para preferencias de enseñanza, mientras que el valor 0 y 10, representan respectivamente el nivel más bajo y alto de preferencia de aprendizaje. A diferencia de las preferencias de enseñanza, las preferencias de aprendizaje no están ligadas a conceptos de dominio sino a un alumno específico. El estado cognitivo de cualquier alumno es inicialmente vacío (es decir $CS(c) = 0$ para cualquier c incluido en un dominio de modelo dado) y puede ser inicializado en un dominio de enseñanza usando un test previo. Las preferencias de aprendizaje se pueden inicializar por el profesor o directamente por los alumnos usando un cuestionario capaz de evaluar los estilos del alumno y transformarlos en valores apropiados para preferencias de aprendizaje. Ambas partes del modelo del alumno se actualizan automáticamente durante las actividades de aprendizaje.

8.1.2.3. El modelo de actividades de aprendizaje

Una *actividad de aprendizaje* debe ser realizada por un alumno para adquirir uno o más conceptos de dominio. Las actividades pueden relacionarse con objetos de aprendizaje (clases textuales, presentaciones, videoclips, podcasts, simulaciones, ejercicios, etc.) o servicios de aprendizaje (laboratorios virtuales, wikis, foros, etc.). El sistema usa actividades de aprendizaje como ladrillos para generar experiencias de aprendizaje. Para usarsede este modo, una actividad de aprendizaje LO se describe a través de los siguientes elementos:

- un conjunto de *conceptos* C_{LO} parte de un modelo de dominio, que son cubiertos por la actividad de aprendizaje.
- un conjunto de *propiedades didácticas* expresadas como una aplicación $DP_{LO}(prop) =$

valor representando estrategias de aprendizaje aplicadas por una actividad de aprendizaje.

- un conjunto de *propiedades de coste* expresadas como una aplicación $CP_{LO}(prop) = valor$ que tenerse en cuenta en el proceso de optimización conectado con el *algoritmo de generación de presentación de aprendizaje.*

Los componentes de las propiedades didácticas tienen el mismo significado con respecto a preferencias de enseñanza y aprendizaje, es decir, *prop* y *valor* pueden tomar valores de un vocabulario cerrado (ver Tabla 8.1). A diferencia de las preferencias de aprendizaje y enseñanza, no están ligadas a un concepto de dominio ni a un estudiante específico sino a una actividad de aprendizaje. Las propiedades de coste son parejas que pueden estar opcionalmente asociadas a actividades de aprendizaje, cuyas propiedades pueden asumir valores del vocabulario cerrado {precio,duración} y cuyos valores son números reales positivos que representan, respectivamente el precio de un único recurso de aprendizaje y su duración en minutos.

8.1.2.4. El modelo de unidad de aprendizaje

Una unidad de aprendizaje representa una secuencia de actividades de aprendizaje necesaria para que un alumno pueda entender un conjunto de conceptos objetivo en un dominio dado con respecto a un conjunto de restricciones de coste [5, 69]. Está compuesta de los siguientes elementos:

- un conjunto de *conceptos objetivo* TC, parte del modelo de dominio, que tienen que ser dominados por un alumno para llevar a cabo la unidad de aprendizaje.
- un conjunto de *restricciones de coste* $CC(prop)=valor$ que deben ser tenidas en cuenta en el proceso de optimización conectado con el algoritmo de generación de presentación de aprendizaje.
- un *camino de aprendizaje* $LPath(c_1, \ldots, c_n)$, es decir, una secuencia ordenada de conceptos que deben enseñarse a un alumno específico para permitirle dominar los conceptos objetivo.
- una *presentación de aprendizaje* $LPres(lo_1, \ldots, lo_m)$, es decir, una secuencia ordenada de actividades de aprendizaje que deben presentarse a un alumno específico para dejarle dominar los conceptos objetivo.

Mientras los conceptos objetivo son definidos por el profesor del curso, el camino de aprendizaje y la presentación de aprendizaje son creadas por algoritmos de generación. Con respecto a los requisitos de coste, la propiedad debe tomar valores del vocabulario cerrado {precio,duración}. Los valores permitidos son números reales positivos que representan, respectivamente, el precio total máximo y la duración total máxima de la unidad de aprendizaje.

La generación del camino de aprendizaje es el punto de partida para generar completamente una unidad de aprendizaje. Empezando por un conjunto de *conceptos objetivo* TC y por un *modelo de dominio*, un camino de conocimiento factible puede ser generado teniendo en cuenta el grafo de conceptos $G(C, BT, IRB, SO)$ parte del modelo de dominio (con $TC \subseteq C$).

Los cuatro pasos del algoritmo de generación del camino de aprendizaje son:

- El *primer paso* construye el grafo $G'(C, BT, IRB', SO')$ propagando relaciones de orden hacia abajo en la relación jerárquica. IRB' y SO' son inicialmente IRB y SO, respectivamente, y a partir de entonces se modifican aplicando la siguiente regla: para cada arco $ab \in IRB' \cup SO'$ sustituir con arcos ac para todos los $c \in C$ tales que existe un camino de c a b en los arcos de BT.

- El *segundo paso* construye el grafo $G''(C', R)$ donde C' es el subconjunto de C que incluye todos los conceptos que deben enseñarse de acuerdo con TC, es decir, C' está compuesto por todos los nodos de G' para los cuales haya un camino ordenado en $BT \cup IRB'$ a concepto en TC (incluyendo los propios conceptos objetivo). R es inicialmente $BT \cup IRB' \cup SO'$ pero todos los arcos referidos a conceptos externos a C' son eliminados.

- El *tercer paso* encuentra una ordenación lineal de nodos de G'' usando una búsqueda en profundidad. La lista obtenida L constituirá la primera aproximación al camino de aprendizaje.

- El *cuarto paso* genera el camino de aprendizaje final $LPath$ eliminando de L todos los conceptos no atómicos con respecto al grafo G, es decir, $LPath$ incluirá cualquier concepto de L menos los conceptos b tales que $ab \in BT$ para algún a. Esto asegura que sólo un concepto hoja será parte del $LPath$.

A continuación es necesario generar, 1) la presentación de aprendizaje apropiada para un alumno específico basándose en un camino de aprendizaje $LPath$ que tiene que ser cubierto, 2) un conjunto de preferencias de enseñanza TP pertenecientes a un modelo de dominio, 3) un estado cognitivo SC y un conjunto de preferencias de aprendizaje LP ambos parte de un modelo de alumno asociado a un alumno objetivo, 4) un conjunto de restricciones de coste

opcionales CC y 5) un conjunto de actividades de aprendizaje. La presentación de aprendizaje se genera siguiendo los siguientes pasos:

- El *primer paso* es seleccionar la sub-lista L de $LPath$ que tiene que convertirse en una presentación. L es la secuencia de todos los conceptos de $LPath$ que todavía no conoce el alumno (es decir, cualquier concepto tal que $CS(a) < \theta$). Si L está vacía entonces el algoritmo termina porque el alumno ya conoce todos los conceptos del camino de aprendizaje.

- El *segundo paso* es definir la mejor secuencia de actividades de aprendizaje P, seleccionadas de las actividades de aprendizaje disponibles, cubriendo L en base a TP, LP y CC. Esta definición de secuencia ha dado el nombre al *Problema de la Presentación de Aprendizaje (PPA)*.

La resolución del PPA define la personalización de la experiencia de aprendizaje. A partir de ahora, PPA será modelado usando el conocido problema PLFN.

Para modelar PPA usando NLFN, lo primero que se debe hacer es definir una medida de distancia $d_{TP}(lo, c)$ entre una actividad lo y el conjunto de preferencias TP con respecto al concepto c. De un modo similar se debe definir una medida de distancia $d_{LP}(lo)$ basada en LP. También se debe definir una nueva medida $d(lo, c)$ como una suma pesada de las otras dos.

Una vez se han definido las medidas de distancia, el problema de seleccionar el mejor conjunto de actividades P que cubran los conceptos de L se convierte en un $NLFN$ que puede esbozarse como el grafo de la Figura 8.2 donde las actividades disponibles se muestran a la izquierda y los conceptos que deben ser cubiertos a la derecha.

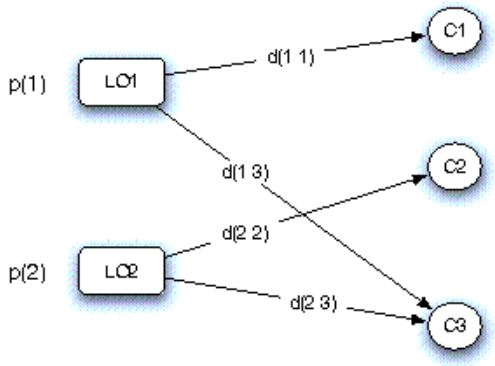


Figura 8.2: Formalización de PPA.

Se puede ver una flecha entre un objeto de aprendizaje LO_i (considerando las actividades disponibles relacionadas con objetos de aprendizaje) y un subconjunto C_j sólo si la instancia de metadatos de LO_i incluye un enlace semántico a los sujetos C_j. Por cada pareja (LO_i, C_j) enlazada en el grafo, hay un valor asignado $d(i,j)$ representando la distancia entre LO_i y C_j. Distancias cortas definen buenas coberturas. Además, a cada objeto de aprendizaje LO_i está asociada a un valor $p(i)$ representando el coste de la introducción del objeto de aprendizaje LO_i en la secuencia de LOs enviados al alumno. Se asume que las distancias $d(i,j)$ se calculan aplicando una función específica que evalúa el emparejamiento entre los valores de metadatos de LO_i que cubren C_j y las preferencias de aprendizaje del alumno que pide la experiencia de e-learning personalizada. Entonces, P debe ser construida como el conjunto más pequeño de actividades que cubren todos los conceptos de L con la mínima suma de distancias entre actividades y conceptos cubiertos. Ahora, se considera m como el número de objetos de aprendizaje y n el número de asignaturas en el camino de aprendizaje que se deben rellenar. Sea y un vector binario donde cada elemento y_i, $(i = 1, \ldots, m)$, supone un valor 1 si se decide usar el objeto de aprendizaje LO_i, y 0 en otro caso; y x como un vector binario en el que cada entrada x_{ij}, $(i = 1, \ldots, m$ y $j = 1, \ldots, n)$, supone un valor 1 si la asignatura C_j se cubre por el objeto de conocimiento LO_i, y 0 en otro caso. Entonces, el modelo del programa de programación lineal que formaliza el problema completo se describe como sigue:

$$\text{mín} \sum_{i=1}^{m} p(i) y_i + \sum_{i=1}^{m} \sum_{j=1}^{n} d(i,j) x_{ij} \tag{8.1}$$

$$s.a: \quad \sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, \ldots, m$$

$$x_{ij} \leq y_i, \quad i = 1, \ldots, m; \quad j = 1, \ldots, n \tag{8.2}$$

$$x_{ij} \in \{0,1\}, \quad i = 1, \ldots, m; \quad j = 1, \ldots, n$$

$$y_i \in \{0,1\}, \quad i = 1, \ldots, m$$

La solución óptima de PPA significa la identificación del conjunto óptimo de objetos de aprendizaje que mejor cumple las preferencias del alumno.

8.2. Construcción de la metaheurística cooperativa

En esta sección se muestran los aspectos importantes en la construcción de la metaheurística cooperativa $\text{KEPS}^2_{CP\ T}$, como son las metaheurísticas que formarán parte de ella, y el

proceso de extracción de conocimiento.

8.2.1. Metaheurísticas integradas

La primera cuestión es la elección de las metaheurísticas que compondrán $\text{KEPS}^2_{\mathcal{CP}\ \mathcal{T}}$. En este caso se han seleccionado cuatro metaheurísticas distintas, dos basadas en poblaciones, Algoritmo Genético y Optimización por Enjambre de Partículas, y dos basadas en trayectorias, Búsqueda Tabú y Temple Simulado. El pseudo-código de estas cuatro metaheurísticas se han presentado en los Algoritmos 12, 13, 14 y 15, del capítulo 6.

8.2.2. Aplicación del proceso de extracción del conocimiento

Para obtener la configuración de $\text{KEPS}^2_{\mathcal{CP}\ \mathcal{T}}$ aplicado a la resolución de PPA es necesario aplicar el proceso de extracción inteligente del conocimiento. Como se explicó previamente, en la primera fase del proceso, la de preparación de datos, se debe resolver un amplio conjunto de instancias para obtener información que pueda ayudar al coordinador a adaptarse a las distintas instancias. En esta situación se utilizan 200 instancias distintas de entrenamiento. De igual manera, cada una de estas instancias debe ser resuelta con cada metaheurística utilizando distintos valores de parámetros. En particular se utilizaron las siguientes configuraciones:

- Algoritmo Genético:
  - Operador de selección: Ruleta y Torneo.
  - Probabilidad de cruce: 0.6 y 0.4.
  - Probabilidad de mutación: 0.01 y 0.001
  - Número de individuos: 100.
- Búsqueda Tabú:
  - Tamaño de la lista tabú: 10, 100 y 1000.
  - Umbral de diversificación: 1, 10 y 50.
- Optimización por Enjambre de Partículas:
  - Número de partículas: 100.
  - Topología: Mejor local considerando 4 vecinos, mejor local considerando 2 vecinos, mejor global y Von Neumann.
  - ω: 0,729844.

- r_p: 1,49 y 2.
- r_g: 1,49 y 2.

- Temple Simulado:
  - Función de enfriamiento: Tangente hiperbólica, coseno hiperbólico, raíz cuadrada y exponencial.
  - Número de iteraciones internas: 5000, 7000, 9000 y 10000.
  - Número de soluciones generadas antes de la selección: 5.

Una vez recopilada toda la información necesaria se obtuvieron los árboles que configuran al agente coordinador de $\text{KEPS}^2_{C'P\ T}$.

8.3. Resultados Experimentales

En esta sección se prueba la validez de $\text{KEPS}^2_{C'P\ T}$ para resolver PPA. Con este objetivo primero se realizan unas pruebas preliminares para mostrar como evoluciona la metaheurística cooperativa comparada con otras metaheurísticas, y finalmente se mostrarán resultados de su aplicación en un entorno real con alumnos y asignaturas reales.

8.3.1. Configuración adicional del sistema

Antes de proceder a mostrar los resultados del sistema es necesario definir algunos parámetros del mismo que se dejaron libres en su descripción, los cuales toman los siguientes valores:

- La medida de rendimiento, ξ, será el valor de la función objetivo 8.1.
- El conjunto $suficiente$ tendrá una función trapezoidal donde $(a, b, c, d) = (0, 0.01, 1, 1)$.
- El α-corte tomará el valor de 0.75.
- El intervalo de frecuencia de intercambio tomará el valor de $[100, 150]$.

8.3.2. Test de rendimiento

Para mostrar gráficamente el rendimiento de $\text{KEPS}^2_{C'P\ T}$ y compararla con otras metaheurísticas se han realizado tests utilizando cuatro tipos de instancias. En estos tests se

compara $\text{KEPS}^2_{\mathcal{CP}\ \mathcal{T}}$ con otras metaheurísticas bicapa no cooperativas que surgen de la combinación secuencial de las metaheurísticas integradas en $\text{KEPS}^2_{\mathcal{CP}\ \mathcal{T}}$. Específicamente, las metaheurísticas bicapa utilizadas son:

- Algoritmo Genético + Búsqueda Tabú (AG+BT).
- Algoritmo Genético + Temple Simulado (AG+TS).
- Optimización por Enjambre de Partículas + Búsqueda Tabú (OEP + BT).
- Optimización por Enjambre de Partículas + Temple Simulado (OEP + TS).

La Figura 8.3 muestra cuatro gráficos que ilustran la evolución de las 5 metaheurísticas. Estos gráficos presentan los resultados medios de 10 ejecuciones utilizando 4000 evaluaciones de la función objetivo, trazando el valor de la función objetivo obtenida por cada metaheurística.

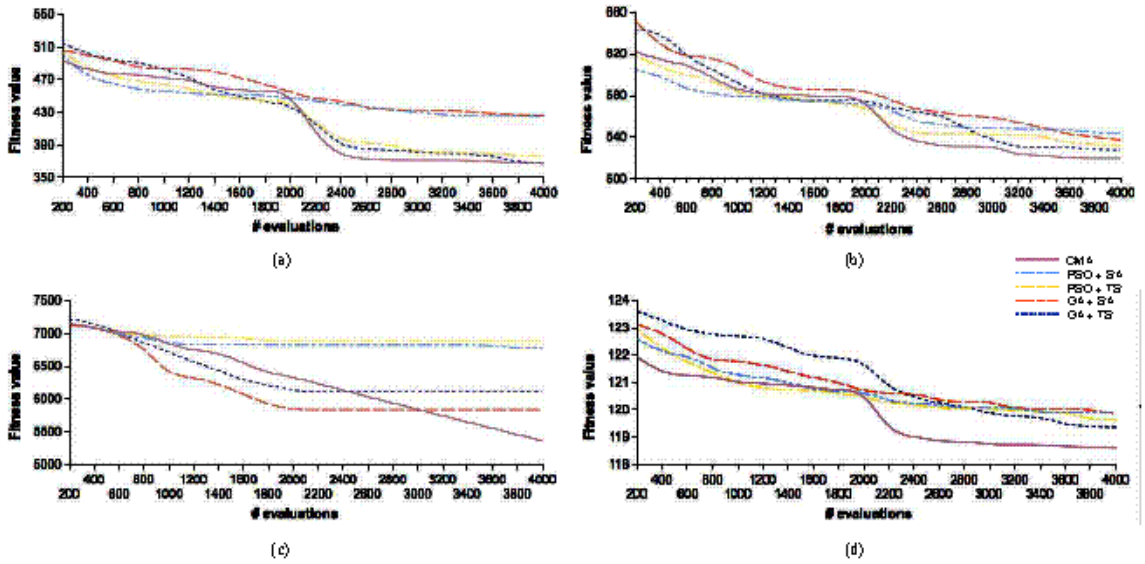


Figura 8.3: Evolución de las distintas metaheurísticas.

De estos gráficos se pueden extraer distintas conclusiones. En el primero (Figura 8.3 (a)), se puede apreciar la robustez de $\text{KEPS}^2_{\mathcal{CP}\ \mathcal{T}}$, puesto que aunque no obtiene los mejores resultados, obtiene un valor de función objetivo que al menos es similar al obtenido por la mejor metaheurística no cooperativa. De hecho, $\text{KEPS}^2_{\mathcal{CP}\ \mathcal{T}}$ es capaz de ignorar los resultados obtenidos por el Temple Simulado (la peor metaheurística para esta instancia). Por otro lado, en la Figura 8.3 (c), todas las metaheurísticas no cooperativas obtienen resultados pobres, mientras que $\text{KEPS}^2_{\mathcal{CP}\ \mathcal{T}}$ obtiene buenas soluciones, lo cual puede ser debido a la

elección inteligente de parámetros que realiza. Finalmente, en las Figuras 8.3 (b) y (d), el rendimiento de todas las metaheurísticas cooperativas es similar pero es mejorado por el obtenido por $\text{KEPS}^2_{\mathcal{CP}\ \mathcal{T}}$. Como resultado de estas pruebas se puede adelantar la robustez de la metaheurística propuesta, que independientemente de que algunos de sus componentes puedan mostrar resultados pobres, es capaz de obtener soluciones de alta calidad y mejores en la mayoría de los casos.

8.3.2.1. Estudios comparativos en un entorno real de e-Learning

Una vez se ha mostrado la robustez del sistema se ha aplicado a un entorno de e-Learning real obteniéndose una importante reducción en el tiempo de aprendizaje de conceptos. En concreto, los tests se han realizado con una experimentación a pequeña escala, en la que participaron 28 alumnos voluntarios pertenecientes a 7 empresas de tamaño pequeño y mediano, que trataban de aprender gestión de empresas. El grupo de alumnos se dividió en dos subgrupos separados: un primer subgrupo compuesto de 20 alumnos utilizó un esquema tradicional sin el uso de $\text{KEPS}^2_{\mathcal{CP}\ \mathcal{T}}$ mientras que un segundo subgrupo, compuesto por 8 alumnos, hizo uso del esquema de aprendizaje usando $\text{KEPS}^2_{\mathcal{CP}\ \mathcal{T}}$. Todos los voluntarios realizaron un test antes y después de realizar el aprendizaje sobre los mismos temas. En todos los tests las capacidades de los alumnos fueron cuantificadas usando tres rangos de habilidad: nivel bajo (puntuación de 0 a 3), nivel medio (puntuación de 4 a 7) y nivel alto (puntuación de 8 a 10). La Figura 8.4 muestra el rendimiento de los dos subgrupos, con y sin el uso de $\text{KEPS}^2_{\mathcal{CP}\ \mathcal{T}}$. Como puede observarse los progresos realizados por los alumnos que utilizaron $\text{KEPS}^2_{\mathcal{CP}\ \mathcal{T}}$ son mucho más acusados que los realizados por los que no utilizaron.

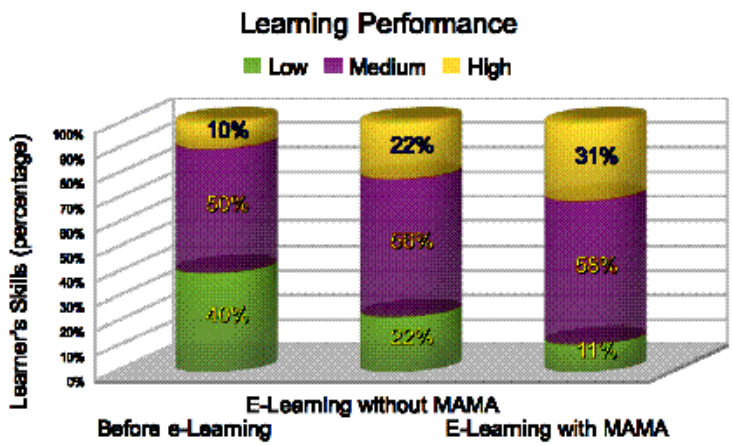


Figura 8.4: Resultados sobre un grupo de 28 alumnos con y sin $\text{KEPS}^2_{\mathcal{CP}\ \mathcal{T}}$

Capítulo 9

Un compositor de música automático

Desde los comienzos de la ciencia de la computación, la generación de artefactos artísticos ha atraído la atención de numerosos investigadores. Una de las artes que más se han beneficiado de este interés es la música, apareciendo numerosos trabajos que demuestran la sinergia existente entre ambas disciplinas. En este capítulo se utiliza $\mathrm{KEPS}^{1}_{\mathcal{CP}\ \mathcal{T}}$ para resolver el problema del bajo continuo, que consiste en generar, de un modo automático, una pieza de música de 4 voces, recibiendo como entrada la línea del bajo (en ocasiones acompañada de indicaciones acerca de que acordes usar).

9.1. El problema del bajo continuo

9.1.1. Introducción

Desde los primeros tiempos de la informática, tanto informáticos como artistas han considerado el uso de ordenadores en la producción de sistemas. La música es una de las artes que más se han beneficiado de la aparición de los ordenadores. La música computacional es tan antigua como los propios ordenadores: la suite de ILLIAC[1] [153, 154] es la primera pieza musical compuesta por un ordenador digital y data del 1957.

El termino "música computacional" abarca varios aspectos en las dos disciplinas, incluyendo programas para notación musical, síntesis de sonido, performances en tiempo real, instrumentos digitales, etc. Este capítulo se centra en la *composición algorítmica*, es decir, el problema de componer música mediante un programa de ordenador sin ninguna intervención humana.

La composición de música algorítmica ha fascinado tanto a informáticos como a músicos. La literatura está llena de trabajos en este sentido, empezando con la suite ILLIAC y llegando

[1] ILLIAC es el nombre de un ordenador construido en los 50.

a esfuerzos más recientes, como, por ejemplo los trabajos propuestos por D. Cope [75–77]. De hecho existen trabajos previos a la era de los ordenadores que pueden ser considerados "algorítmicos" (como por ejemplo el juego de dados de Mozart).

Se han usado varias herramientas para obtener compositores algorítmicos: números aleatorios, gramáticas formales, autómatas celulares, fractales, inteligencia artificial, algoritmos evolutivos, música genética (música basada en ADN y proteínas). El libro de Miranda [193] contiene una buena revisión.

La música occidental, a partir del llamado periodo de la práctica común, está basada en el sistema de música temperamento igual. Este sistema de música es en el que se escribe toda la música occidental y está basada en reglas ampliamente aceptadas. El problema que se tratará es uno muy específico, conocido como el problema del bajo continuo. Este problema surgió en el siglo XVI aunque encontró su auge durante el barroco. En este problema, el compositor solamente crea la voz del bajo, pero no especifica el contrapunto o los acordes del ripieno, es decir, el resto de voces, que deja a cargo del interprete, el cual debe improvisar aquellos que mejor se acomodan a la armonía del conjunto. En ocasiones, la composición incluye *cifras* o indicaciones para guiar al intérprete en la elección de los acordes, en este caso el problema se conoce como problema del bajo continuo cifrado. Por tanto, en este problema se tiene una línea de bajo como entrada y el objetivo es componer las otras 3 voces para completar una pieza completa de música a 4 voces. Ejemplos inalcanzables de estas composiciones son los corales de Bach.

A menudo, los algoritmos son usados para encontrar un óptimo o soluciones buenas, donde óptimo y bueno están definidos de acuerdo a una métrica definida precisamente. Para composiciones musicales esta métrica precisa no existe. Esto es un serio problema porque no existe una forma fácil de decir que una solución es "mejor" que otra. La bondad de una composición musical (como cualquier otra producción artística) es una opinión subjetiva. En el contexto del problema que estamos considerando, las reglas armónicas bien conocidas son de ayuda, aunque no constituyen una métrica.

La existencia de una métrica precisa reduciría el problema de composición de música a un problema de búsqueda en un espacio de búsqueda muy grande: la composición de música se podría ver como un problema combinatorio que consiste en colocar un número finito de notas en un pentagrama finito. Desafortunadamente (¿o afortunadamente?), como se ha dicho, esta métrica no existe. Para "comparar" soluciones usamos reglas de armonía.

Como se ha comentado anteriormente, existen muchas herramientas y técnicas que pueden usarse para diseñar un compositor algorítmico. Las metaheurísticas son una de ellas. En este capítulo se utilizará $\text{KEPS}^{1}_{\mathcal{CP}\ \ \mathcal{T}}$ para resolver el problema del bajo continuo.

9.1.2. Antecedentes musicales

Como se comentó con anterioridad, se considera el sistema musical de temperamento igual usado en los países occidentales. Este sistema, define octavas empezando por un sonido de referencia (una frecuencia dada, normalmente 440Hz) que son aquellos sonidos cuyas frecuencias pueden obtenerse repetidamente doblando/reduciendo a la mitad la frecuencia de referencia. Cada octava está dividida en 12 notas espaciadas igualmente[2] , denotadas por las letras A, A# o B♭, C, C# o D♭, D, D# o E♭, E, F, F# o G♭, G y G# o A♭. La música de temperamento igual está basada en la noción de tonalidad. En un sentido poco estricto, una tonalidad es un grupo de notas que forman una escala. Empezando por cada una de las 12 notas de una octava se puede tener una tonalidad (hay varios tipos para cada nota, como mayor, menor, etc.). Por ejemplo, la escala mayor de C es C, D, E, F, G, A, B, mientras que la escala mayor de D es D, E, F#, G, A, B, C#. Las notas de una escala se denotan a menudo también como I, II, III, IV, V, VI, VII y VIII especialmente cuando se quiere enfatizar sólo el grado de la escala y no la nota particular, que depende de la tonalidad. Una pieza escrita en una tonalidad dada puede ser fácilmente transpuesta a otra tonalidad moviendo simplemente todas las notas, esto es posible puesto que todas las notas están espaciadas igualmente. Una buena explicación del la música de temperamento igual se puede encontrar en [181].

Usualmente una tonalidad es considerada la tonalidad principal de una pieza, y por tanto las notas de la escala correspondiente se consideran más importantes que las notas fuera de la misma. La música occidental, empezando por el periodo de la práctica común está basada en reglas armónicas y melódicas bien establecidas para el sistema de temperamento igual. Para una discusión acerca de estas reglas se puede leer el libro de armonía [217].

Nuestro interés se centra en la música compuesta para 4 voces, como los corales de Bach. Un coral está compuesto de 4 voces independientes, llamadas *bajo*, *tenor*, *contralto* y *soprano*. Una pieza de música consiste en una secuencia de *compases*, cada uno consistente en un número de unidades de tiempo dadas. En cada unidad de tiempo las cuatro voces interpretan una nota[3] . El conjunto vertical de notas en una unidad de tiempo es un *acorde*. La noción de acorde es fundamental y hay numerosos tipos de acorde. En este trabajo se consideran sólo aquellos compuestos de 3 y 4 notas. Los acordes se construyen sobre cada grado de la escala, esto es I, II, III, IV, V, VI, VII, VIII. A menudo la nota sobre la que se construye el acorde, llamada *nota raíz* del acorde, es dada al bajo; sin embargo el bajo puede interpretar cualquier otra nota (inversiones de acorde). Las inversiones de acorde son denotadas usualmente con un superíndice que indica la inversión (por ejemplo, III^6, V^{46}, I^{357}).

[2] Para ser precisos si una nota tiene frecuencia f la "siguiente" nota tiene frecuencia $f \cdot 2^{\frac{1}{12}}$.

[3] Esto es una simplificación de lo que sucede en realidad, puesto que una composición puede tener también notas de paso que no son parte de la armonía

Los acordes considerados se muestran en la tabla 9.1 (para cada uno se da un ejemplo en la tonalidad de C).

Tabla 9.1: Acordes (con ejemplos en la tonalidad de C).

Acorde	Nota raíz	Conjunto de notas
Triada mayor	C	C, E, G
Triada menor	E	E, G, B
Séptima mayor	C	C, E, G, B
Séptima menor	D	D, F, A, C
Séptima dominante	G	G, B, D, F
Séptima semidisminuida	B	B, D, F, A

Una de las reglas fundamentales del sistema de música de temperamento igual es que el conjunto vertical de notas debe ser uno de los acordes permitidos. Otras reglas afectan a secuencias de acordes. Algunas secuencias son "mejores" que otras, donde mejores es difícil de definir puesto que es una evaluación subjetiva. En cualquier modo, está ampliamente aceptado que secuencias particulares de acordes funcionan mejor que otras. Algunos acordes son "más importantes" que otros porque sugieren, preparan o refuerzan centros tonales. El arte de la música tonal consiste precisamente en colocar los acordes de tal manera que su interacción sea agradable y significativa. Del lado práctico, esto se traduce en reglas simples que indican cosas como (tomada de [217]):

I es seguido de IV o V, algunas veces VI, menos a menudo II o III

II es seguido de V, algunas veces de VI, menos a menudo I,III o IV

Además de reglas acerca de secuencias de acordes, también existen reglas acerca de líneas melódicas. Una línea melódica es simplemente la secuencia de notas tocadas por cada voz. Las reglas acerca de líneas melódicas se pueden referir al movimiento de una sola voz (por ejemplo, normalmente un salto mayor que una octava no está permitido) o también a los movimientos de dos voces (por ejemplo dos voces, que continuan con una quinta paralela no están permitidas).

Una descripción detallada se puede encontrar en el libro estándar de armonía [217].

9.1.3. Proceso de resolución del problema

El problema del bajo continuo se puede resolver en dos fases diferentes. En la primera se deben obtener las cifras que guíen en la elección de los acordes, o lo que es lo mismo se debe pasar del problema del bajo continuo al del bajo continuo cifrado. Para esta fase se utilizará un algoritmo basado en programación dinámica propuesto en [93]. Para la segunda fase, que trata de resolver el problema del bajo continuo cifrado se usará $\text{KEPS}^1_{CP\ T}$.

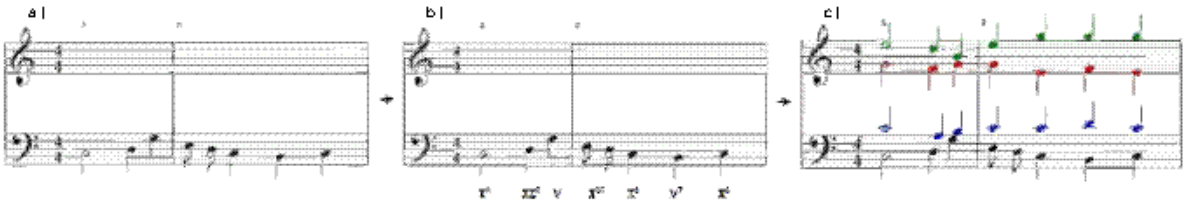


Figura 9.1: Ejemplo del problema

En la Figura 9.1 se presenta un ejemplo del problema y una posible solución. La parte *a)* de la figura proporciona la entrada - sólo una línea de bajo; la parte *b)* muestra la misma línea pero con indicaciones acerca de los acordes a usar, mientras que la parte *c)* proporciona una solución.

9.1.4. Evaluación de la solución

Como se explicó con anterioridad, la función de evaluación de la solución utilizará reglas de composición musical para valorar como de buena es una solución. Actualmente se consideran las reglas representadas en la Tabla 9.2.

Para asignar un valor a una solución se empieza de un valor base asignado a la secuencia de acordes obtenido en la primera fase de la resolución, [93]. A continuación se resta un coste dado por cada regla que se cumpla. Las relaciones entre dos acordes consecutivos nos permiten decidir si una solución es buena o no. Por ejemplo, si una voz realiza un salto de una cuarta aumentada, entonces se resta 40 al valor inicial, si dos voces hacen una quinta oculta, se resta 1000. La Tabla 9.2 también muestra el costo que se usa para cada regla. Se debe tener en cuenta que los acordes posibles se construyen siempre poniendo las voces en el orden: bajo, tenor, alto y soprano, por tanto, no se permiten cruces de voces.

Tabla 9.2: Reglas de valoración de la solución

Errores de una voz	Coste	tipo	Errores de dos voces	Coste	Tipo
sexta	1000	normal	unisono oculto	100	normal
cuarta aumentada	1000	normal	quinta oculta	100	normal
quinta aumentada	1000	normal	octava oculta	100	normal
séptima	3000	critico	unisono	3000	critico
>octava	3000	critico	quinta paralela	3000	critico
			octava paralela	3000	critico
Coste de una voz	Coste	tipo	Errores en el acorde	Coste	Tipo
salto	intervalo	no es error	salto de una octava	100	normal

9.1.5. Trabajos relacionados

El diseño de compositores automáticos capaces de generar la armonía de 4 voces ya ha sido considerado en algunos artículos. Por ejemplo, en [108] se describe un compositor automático basado en reglas y sistemas expertos. Otros ejemplos son EMI, propuesto en [75,77], que utiliza una combinación de varias técnicas (gramáticas formales, reglas, análisis musical, pattern matching), o el propuesto en [122] se utiliza una aproximación basada en redes neuronales.

Ninguna de las aproximaciones expuestas anteriormente hace uso de metaheurísticas, sin embargo, también han existido algunos esfuerzos por aplicarlas a la composición automática, principalmente centrados en los algoritmos genéticos. Entre estas aproximaciones se pueden citar [27] para la generación de solos de Jazz, [158] que utiliza un algoritmo genético interactivo (es decir, que necesita intervención humana) o [155] que armoniza progresiones de acordes. Otros artículos que utilizan algoritmos genéticos y que resuelven un problema más parecido al que se trata en este capítulo son [186, 259] que es la armonización de una línea melódica.

9.2. Construcción de la metaheurística cooperativa

En esta sección se explican aspectos importantes en la construcción de la metaheurística cooperativa, como son las metaheurísticas que formarán parte de ella, y el proceso de extracción de conocimiento.

9.2.1. Metaheurísticas integradas

La primera cuestión a resolver es la elección de las metaheurísticas que compondrán $\text{KEPS}^1_{\mathcal{CP}\ \mathcal{T}}$. Para este caso se han seleccionado cuatro metaheurísticas distintas, dos basadas en poblaciones, Algoritmo Genético y Optimización por Enjambre de Partículas, y dos basadas en trayectorias, Búsqueda Tabú y Temple Simulado. El pseudo-código de estas cuatro

metaheurísticas se han presentado en los Algoritmos 12, 13, 14 y 15, del capítulo 6.

9.2.2. Aplicación del proceso de extracción del conocimiento

Para obtener la configuración de $\text{KEPS}^{1}_{C^{P}\ T}$ aplicado a la resolución del problema del bajo no continuo cifrado es necesario aplicar el proceso de extracción inteligente del conocimiento. Como se explicó previamente, en la fase preparación de datos, se debe resolver distintas instancias para obtener información que pueda ayudar al coordinador a adaptarse a las posibles instancias. En el caso que atañe a este capítulo se utilizaron 36 instancias distintas de entrenamiento. De igual manera, cada una de estas instancias debe ser resuelta con cada metaheurística utilizando distintos valores de parámetros. En particular se utilizaron las siguientes configuraciones:

- Algoritmo Genético:
  - Operador de selección: Ruleta y Torneo.
  - Probabilidad de cruce: 1, 0.9, 0.8 y 0.5.
  - Probabilidad de mutación: 0.3, 0.15, 0.1 y 0.01.
  - Número de individuos: 20, 30 y 40.
- Búsqueda Tabú:
  - Tamaño de la lista tabú: 2, 4, 5, 10 y 20.
  - Umbral de diversificación: 5, 10, 15, 100 y 200.
- Optimización por Enjambre de Partículas:
  - Número de partículas: 20, 30 y 40.
  - Topología: Mejor local considerando 4 vecinos, mejor local considerando 6 vecinos, mejor global y Von Neumann.
  - ω: 0.6, 0,729844, 0.8.
  - r_p: 1.1, 1.2, 1,49 y 1.6.
  - r_g: 1.4, 1.49, 1.6 y 1.8.
- Temple Simulado:
  - Función de enfriamiento: Tangente hiperbólica, coseno hiperbólico, raíz cuadrada y exponencial.

- Número de iteraciones internas: 10, 100, 1000.
- Número de soluciones generadas antes de la selección: 10, 15, 30 y 50.

Una vez recopilada toda la información necesaria se obtuvieron los árboles que configuran al agente coordinador de $\text{KEPS}^1_{C^P\ T}$.

9.3. Resultados Computacionales

En esta sección se probará la validez de la estrategia propuesta para resolver el problema del bajo no continuo. Con este objetivo primero se realizarán unas pruebas preliminares para mostrar como resuelve el problema, y a continuación se compararán sus resultados con los obtenidos por una metaheurística aparecida en la literatura [93].

9.3.1. Configuración adicional de la metaheurística cooperativa

Antes de proceder a mostrar los resultados del sistema es necesario definir algunos parámetros del mismo que se dejaron libres en su descripción, los cuales toman los siguientes valores:

- La medida de rendimiento, ξ, será el valor de la función objetivo.
- El conjunto $suficiente$ tendrá una función trapezoidal donde $(a, b, c, d) = (0, 0.01, 1, 1)$.
- El α-corte tomará el valor de 0.001.
- El intervalo de frecuencia de intercambio tomará el valor de $[5000, 5500]$.

9.3.2. Resultados preliminares

Se han ejecutado varias pruebas con distintas instancias diferentes de las utilizadas en el proceso de extracción del conocimiento. La Figura 9.2 muestra un ejemplo de los resultados obtenidos por la técnica, al aplicarla al coral BWV 266 de J.S. Bach. La parte *a*) de la figura muestra la línea del bajo (entrada), la parte *b*) muestra una solución generada en los primeros pasos de la ejecución y la parte *c*) muestra la solución final encontrada.

Los resultados obtenidos cumplen las expectativas, devolviendo soluciones de una calidad aceptable y en ningún caso discordantes. Esto es así porque, ayudada por las reglas que definen la función objetivo, la metaheurística cooperativa es capaz de eliminar prácticamente todos los errores. En particular, en la solución mostrada se han eliminado la mayoría de errores.

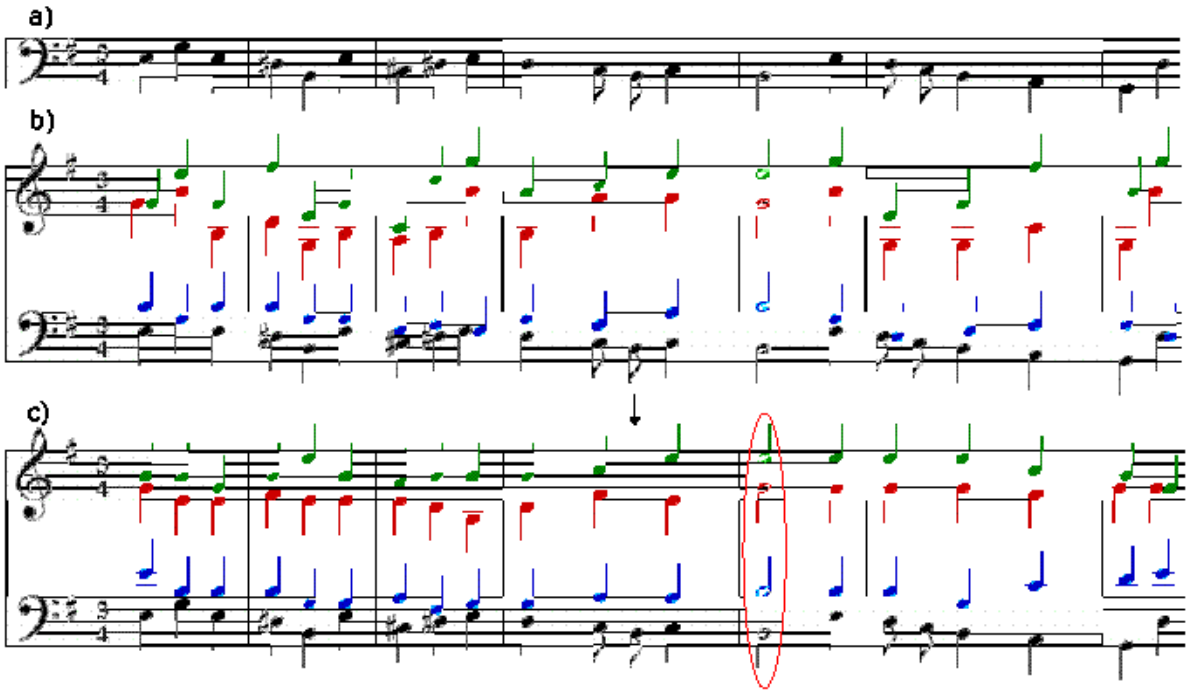


Figura 9.2: Un ejemplo de entrada (parte a), y solución final (parte b)

En la parte de la solución que se muestra en la figura se puede ver uno de los errores que no se han eliminado. El error está localizado en el quinto compás y resaltado por una elipse roja. Este tipo de error se conoce como octavas paralelas y se produce cuando dos voces están separadas por un intervalo de octava, y se mueven en la misma dirección terminando de nuevo con la misma separación. Para preservar la independencia de las voces se evitan las octavas paralelas, puesto que su utilización anula una de las voces.

Por último resaltar que el tiempo utilizado en la generación de las soluciones es pequeño, aproximadamente un minuto, lo que permite al decisor repetir varias veces el proceso pudiendo seleccionar aquella solución que más le guste. Esto es importante puesto que, como se resaltó en la introducción, la bondad de una composición musical es una opinión subjetiva.

9.3.3. Comparación con Evolutionary Music Composer [93]

En esta subsección se compara el rendimiento obtenido por $\text{KEPS}^1_{\mathcal{CP}\ \mathcal{T}}$ con el obtenido por una metaheurísticas recientemente publicada, Evolutionary Music Composer (EMC) [93]. EMC utiliza un algoritmo genético especialmente diseñado para resolver el problema del bajo continuo, en el que el cruce y la mutación están especializados con la finalidad de reducir el número de errores de armonía que aparecen en la solución.

La Tabla 9.3 muestra los resultados de ambas aproximaciones. En cada celda se muestra la media del error cometido, con respecto a la mejor solución encontrada hasta el momento, después de 10 ejecuciones. Cada metaheurística se ejecutó hasta que los resultados se estabilizaron.

Tabla 9.3: Comparativa EMC vs. $KEPS^1_{CP\ \ T}$

instancia	EMC	$KEPS^1_{CP-T}$
Coral de Bach n°26	$0,10_{0,02}$	$0,01_{0,01}$
Coral de Bach n°70	$0,06_{0,02}$	$0,02_{0,02}$
Coral de Bach n°112	$0,10_{0,01}$	$0,03_{0,03}$
Coral de Bach n°215	$0,04_{0,03}$	$0,03_{0,02}$
Coral de Bach n°259	$0,07_{0,02}$	$0,03_{0,02}$
Coral de Bach n°266	$0,12_{0,01}$	$0,02_{0,02}$
Coral de Bach n°375	$0,06_{0,02}$	$0,02_{0,01}$
Coral de Bach n°606	$0,11_{0,02}$	$0,02_{0,02}$
Coral de Bach n°2011	$0,04_{0,02}$	$0,04_{0,03}$
Coral de Bach n°3604	$0,02_{0,01}$	$0,03_{0,03}$
Coral de Bach n°14403	$0,10_{0,01}$	$0,13_{0,25}$
Coral de Bach n°15309	$0,11_{0,01}$	$0,02_{0,01}$

Como puede observarse los resultados en cuanto a fitness son muy similares, aunque en todos los casos menos en uno $KEPS^1_{CP\ \ T}$ obtuvo mejores resultados que EMC. De hecho, las mejores soluciones, con las que se calcula el error, fueron siempre encontradas por $KEPS^1_{CP\ \ T}$. Además, consultando a un experto en música, afirmó que las soluciones encontradas por $KEPS^1_{CP\ \ T}$ tienen un sonido de mayor calidad que las obtenidas por EMC. Por todo ello, se puede concluir que $KEPS^1_{CP\ \ T}$ es una metaheurística cooperativa interesante para resolver este problema.

Capítulo 10

Un problema de localización

En este capítulo se probará la efectividad de la metaheurística cooperativa con un problema de localización, concretamente el Uncapacited Single Allocation p-Hub Median Problem. Los problemas de localización comprenden una importante área dentro de los problemas de optimización y por ello es interesante comprobar como se comporta nuestro meteheurística cooperativa en ellos. Además, el problema que se tratará tiene la peculiaridad de que existen muy pocas instancias de él, lo que puede ser un problema para el proceso de extracción del conocimiento, y por tanto, se podrá comprobar como se comporta la metaheurística ante este tipo de problemas.

10.1. Los problemas de localización

10.1.1. Breve introducción histórica

La literatura acerca de los problemas de localización, en su versión puramente matemática, es antigua y ha sido recopilada por diversos autores. En el siglo XVII, Fermat planteó una cuestión que puede considerarse el germen de los problemas de localización posteriores: dados tres puntos en el plano, ¿cuál sería la localización de un cuarto punto que cumpliese que la suma de sus distancias a los tres puntos dados sea mínima? A partir de 1600, varios autores propusieron distintos enfoques para resolver dicho problema.

A pesar de estas tempranas aventuras matemáticas, no fue sino hasta principios del siglo XX cuando Weber [255] formuló un modelo de localización que fué la base de toda la abundante literatura posterior sobre localización económica.

En los años siguientes a la publicación del trabajo de Weber hubo pocas aportaciones a la teoría de la localización. Por una parte, Isard [157] y Predöhl [220] desarrollaron las ideas de Weber. Mientras que Cooper [74], por su parte, planteó y estudió un nuevo modelo que sería

llamado posteriormente el problema de localización de plantas sin restricciones de capacidad.

10.1.2. Elementos de un problema de localización

El campo abarcado por la teoría de la localización es tan amplio que necesariamente debe de haber un gran número de modelos, cada uno con sus propias características. Encontrar una definición rigurosa que abarque a todos estos modelos se puede calificar de casi imposible y, por ello, hay pocas definiciones genéricas y, aun éstas, son tan poco descriptivas como la de Brandeau y Chiu [36] quienes afirman que "un problema de localización es un problema espacial de asignación de recursos".

Mucho más útil que tratar de definir el concepto "problema de localización", es el estudio de los elementos comunes en los modelos de localización que luego nos ayudarán a comprender el funcionamiento de un problema concreto.

10.1.2.1. Estructura topológica

Puesto que los problemas de localización se plantean en un espacio geográfico, necesitamos una estructura topológica que lo represente adecuadamente, a la que nos referiremos como la estructura topológica subyacente del problema. Muy pronto se empezaron a distinguir dos categorías: los problemas sobre un plano y los problemas sobre una red. Por último, algunos autores, como Batta y Palekar [18], consideran problemas híbridos red/plano en los que parte del espacio topológico subyacente es un espacio n-dimensional mientras otras áreas pueden modelarse como una red.

10.1.2.2. Función de distancia

Todos los problemas de localización necesitan una función de distancia o métrica que mida la ubicación de los elementos relevantes del problema.

10.1.2.3. Demanda

Las dos principales características de la demanda que influyen en la formulación de un problema de localización son su elasticidad y su agregación. Cuando en un modelo la demanda viene dada, y no depende del nivel de servicio, decimos que es un problema de localización con demanda inelástica. En caso contrario, se trata de un problema con demanda elástica. Por otra parte, cuando la demanda se considera agregada, se habla de "puntos demanda" para denotar su ubicación, y de clientes o consumidores para referirse a cada uno de los datos que

se han agregado.

10.1.2.4. Centros de servicio

En el vocabulario de la teoría de la localización, un centro de servicio es aquella instalación que se debe localizar para atender la demanda. Es obvio que las características asociadas a los centros de servicio afectan mucho a la formulación y resolución del problema asociado. Por eso es usual distinguir el problema por aquella característica que más le influye.

De este modo, se habla de problemas:

- con uno o con múltiples centros, según la cantidad que haya que localizar;
- si permitimos que cada centro pueda ofrecer diferentes servicios distinguibles, estamos ante un problema de localización multiproducto como opuesto a los problemas uniproducto;
- dependiendo del sentimiento que despierten en los consumidores, distinguimos centros deseables o amigables, semideseables o seminocivos, y nocivos o desagradables;
- y podríamos seguir enumerando problemas con centros que interactúan (o no) entre sí, con o sin restricciones de capacidad, etc.

10.1.3. Clasificación de los problemas de localización

Podemos distinguir entre localización en planos, en redes e híbrida o entre problemas con centros agradables, semideseables y nocivos. Algunas más del mismo tipo serían la distinción entre localización uniobjetivo cuando se emplea un único criterio de optimización, multiobjetivo cuando hay más de un criterio de optimización, o sin criterio de optimización; localización determinista cuando todos los datos del problema son conocidos con exactitud o localización no determinista en caso contrario; localización estática cuando los datos del problema no dependen del tiempo o localización no estática en caso contrario; etc.

En cada caso concreto, es usual clasificar el problema bajo estudio con algunas de ellas referidas al aspecto o aspectos que se quieren destacar. Y así, encontramos trabajos en cuyos títulos aparecen expresiones como localización de centros de servicio públicos, localización de centros de servicio sin restricciones de capacidad o análisis multiobjetivo de decisiones de localización.

Otra práctica habitual consiste en utilizar el nombre del problema prototipo en el que se basa el problema estudiado, añadiendo algún comentario que identifique la hipótesis o las

hipótesis que se han cambiado. Por ejemplo, podemos encontrar el problema de Weber sobre una esfera, el problema de localización de plantas con restricciones de capacidad o el problema de la p-mediana restringido por zonas.

10.2. Problemas de tipo Hub

Un tipo específico de problemas de localización son los problemas de tipo hub. Estos problemas aparecen cuando es necesario guiar el flujo de un origen a un destino, pero no es recomendable, o es demasiado caro, crear conexiones entre cada par origen-destino. El objetivo de este tipo de problemas está compuesto por dos pasos:

- Localización de los hubs: para determinar que nodos deben ser hubs y su número, con el objetivo de distribuir el flujo a través de ellos.
- Asignación de no-hubs a hubs: para asignar el resto de nodos a los hubs.

Generalmente, estas tareas se realizan minimizando una función objetivo que describe el flujo de intercambio y su coste. Se puede encontrar una revisión general de este tipo de problemas en [66].

10.3. The Uncapacited Single Allocation p-Hub Median Problem

En este capítulo nos centraremos en el Uncapacited Single Allocation p-Hub Median Problem (USApHMP). Uncapacited (no capacitado) significa que los hubs no tienen restricción acerca de la cantidad de flujo que pueden manejar, single (individual) indica que los nodos sólo se pueden asignar a un hub, y p-Hub significa que el número de hubs debe ser p.

En este capítulo se utilizará la formulación dada por O'Kelly en [206]. Sea N un conjunto de n nodos. Se define W_{ij} como el flujo existente desde el nodo i al j, y C_{ij} como el coste de transporte unitario entre los nodos i y j.

Sea X_{ij} una variable de decisión definida como:

$$X_{ij} = \begin{cases} 1 & \text{si nodo } i \text{ es asignado al hub } j. \\ 0 & \text{en otro caso} \end{cases}$$

El USApHMP se puede formular como:

$$min \sum_{i,j,k,l \in N} W_{ij}(\xi C_{ik} X_{ik} + \alpha C_{kl} X_{ik} X_{jl} + \delta C_{jl} X_{jl}) \tag{10.1}$$

$$s.a$$

$$\sum_{j=1}^{n} X_{jj} = p \tag{10.2}$$

$$\sum_{j=1}^{n} X_{ij} = 1, \quad \forall i = 1, \ldots, n \tag{10.3}$$

$$X_{ij} \leq X_{jj}, \quad \forall i, j = 1, \ldots, n \tag{10.4}$$

$$X_{ij} \in \{0, 1\}, \quad \forall i, j = 1, \ldots, n \tag{10.5}$$

Los parámetros ξ, α y δ representan, respectivamente, el coste de recogida (generalmente $\xi = 1$), el coste de transferencia (generalmente $\alpha < 1$) y el coste de distribución (generalmente $\delta = 1$). La función objetivo 10.1 minimiza la suma de los costes de flujo hub-origen, hub-hub y hub-destino. La restricción 10.2 asegura que se utilicen exactamente p hubs. 10.3 indica que un nodo sólo puede ser asignado a un hub. La condición 10.4 garantiza que un punto que no sea hub sólo puede ser asignado a un hub y no a otro nodo no hub. Finalmente, 10.5 es la clásica restricción binaria.

El USApMP es un problema NP-duro. Además, a pesar de que el conjunto de hubs es fijado de antemano, el sub-problema de asignación también es NP-duro [180].

Las instancias elegidas para la experimentación se obtuvieron de la librería ORLIB [22]. Concretamente, se utilizó el conjunto de datos AP (Australian Post) derivado de un estudio del sistema postal australiano. Las instancias utilizadas se pueden dividir en dos grupos, aquellas con 50 nodos o menos y aquellas con más de 50 nodos. Para el primer conjunto se consideraron instancias con 2, 3, 4 y 5 hubs, mientras que para el segundo el número de hubs puede tomar los valores: 2, 3, 4, 5, 10, 15 y 20. El valor de las constantes ξ, α y δ es fijado a 3, 0.75 y 2 respectivamente. El óptimo para aquellas instancias con un número de nodos menor de 50 fue obtenido de ORLIB, y para el resto de instancias se considera la mejor solución obtenida hasta el momento presentada en [169].

10.4. Construcción de la metaheurística cooperativa

En esta sección se explican aspectos importantes en la construcción de la metaheurística cooperativa, como son las metaheurísticas que formarán parte de ella, y el proceso de extracción de conocimiento.

10.4.1. Metaheurísticas integradas

La primera cuestión a resolver es la elección de las metaheurísticas que compondrán $\text{KEPS}^{1}_{C\rho\ T}$, para este caso se han seleccionado tres metaheurísticas distintas, todas basadas en trayectorias: Búsqueda Tabú, Temple Simulado y Búsqueda Descendente por Entornos Variables. El pseudo-código de las metaheurísticas Búsqueda Tabú y Temple Simulado se ha presentado en los Algoritmos 13 y 14 del capítulo 6. A continuación se mostrará la metaheurística de Búsqueda Descendente por Entornos Variables.

10.4.1.1. Búsqueda por Entornos Variables

El pseudo-código de la Búsqueda se presenta en el algoritmo 16.

Algoritmo 16: Pseudo-código de la Búsqueda por Entornos Variables

Input: q: instancia del problema, $\mathcal{N} = \{\mathcal{N}_i\}$, $i = 1, \ldots, k_{max}$: conjunto de posibles vecindarios
Output: *solucion*: mejor solución encontrada
begin
  s_a=solución-inicial(q);
  repeat
    k=1;
    s_{mejor}=s_a;
    repeat
      s_v = ObtenerMejorVecino($\mathcal{N}_k(s_a)$);
      if $f(s_v) < f(s_a)$ **then**
        $s_a = s_v$;
      end
      $k = k + 1$;
    until $k = k_{max}$;
    if $f(s_a) \geq f(s_{mejor})$ **then**
      finalización=T;
    end
  until *finalizacion*;
  devolver s_{mejor}
end

10.4.2. Aplicación del proceso de extracción del conocimiento

Para obtener la configuración de $\text{KEPS}^{1}_{C\rho\ T}$ aplicado a la resolución del USApMP es necesario aplicar el proceso de extracción inteligente del conocimiento. Sin embargo, en el

caso del USApMP se debe tener en cuenta que existe un número de instancias muy reducido, 34 o, lo que es lo mismo, una instancia de cada tipo posible. Esto dificulta el proceso de extracción inteligente de conocimiento, puesto que es más difícil la obtención de información precisa. Del mismo modo también dificulta el proceso de prueba, ya que, para no adulterar los resultados, es necesario utilizar instancias de prueba distintas de las que se usaron en el aprendizaje. Por ello, en el caso de este problema fue necesario modificar ligeramente los procesos de extracción del conocimiento y prueba, siguiendo una aproximación similar a la utilizada en la validación cruzada. La validación cruzada, consiste en dividir una muestra de datos en subconjuntos de tal modo que el análisis inicialmente se realiza en uno de ellos, mientras los otros subconjuntos son retenidos para su uso posterior en la confirmación y validación del análisis inicial. De esta manera, el conjunto de datos se dividirá en dos partes, una que se usará para realizar el aprendizaje y otra para realizar las pruebas, repitiéndose este proceso varias veces. El resto del proceso de extracción del conocimiento y el de prueba se mantienen invariables.

Así, en la preparación de datos, se deben resolver las instancias de entrenamiento con cada metaheurística utilizando distintos valores de parámetros, recopilando la información interesante. Una vez recopilada toda la información necesaria se obtienen los árboles que configuran al agente coordinador de $\text{KEPS}^1_{\mathcal{CP}\ \mathcal{T}}$.

10.5. Experimentos y resultados

Los experimentos de este capítulo tienen por objetivo comprobar la problemática que surge cuando se disponen de pocas instancias de aprendizaje, y los beneficios que puede aportar el proceso de extracción del conocimiento en estos casos. De esta manera se considerarán las siguientes metaheurísticas:

- $\text{KEPS}^1_{\mathcal{M}em}$.
- $\text{KEPS}^1_{\mathcal{M}em\ \mathcal{T}}$: metaheurísticas similar a $\text{KEPS}^1_{\mathcal{M}em}$ en la que la elección de los parámetros de cada metaheurística se realiza utilizando modelos obtenidos con el proceso de extracción inteligente del conocimiento.
- $\text{KEPS}^1_{\mathcal{CP}\ \mathcal{TPF}}$: metaheurísticas similar a $\text{KEPS}^1_{\mathcal{CP}\ \mathcal{T}}$ en la que la elección de los parámetros de cada metaheurística se fija para todas las instancias y se utilizan valores de parámetros obtenidos del conocimiento experto.
- $\text{KEPS}^1_{\mathcal{CP}\ \mathcal{T}}$.

Comparando estas metaheurísticas se pretende comprobar, por un lado, la influencia que tiene la regla de intercambio de soluciones utilizada, ya sea la basada en memoria o la basada en conocimiento, así como la influencia que tiene el modo de configurar los parámetros, con conocimiento experto y estático o con conocimiento obtenido a través de un proceso de extracción inteligente.

Antes de proceder a mostrar los resultados es necesario definir algunos parámetros del mismo que se dejaron libres en su descripción, los cuales toman los siguientes valores:

- La medida de rendimiento será el valor de la función objetivo 10.1.
- El conjunto *suficiente* tendrá una función trapezoidal donde $(a, b, c, d) = (0, 0.01, 1, 1)$.
- El α-corte tomará el valor de 0.75.
- El intervalo de frecuencia de intercambio tomará el valor de $[100, 150]$.

Además cada instancia se resolvió 20 veces.

10.5.1. Influencia de la regla de intercambio de soluciones

La Figura 10.1 muestra dos diagramas de dispersión en los que se comparan las distintas metaheurísticas.

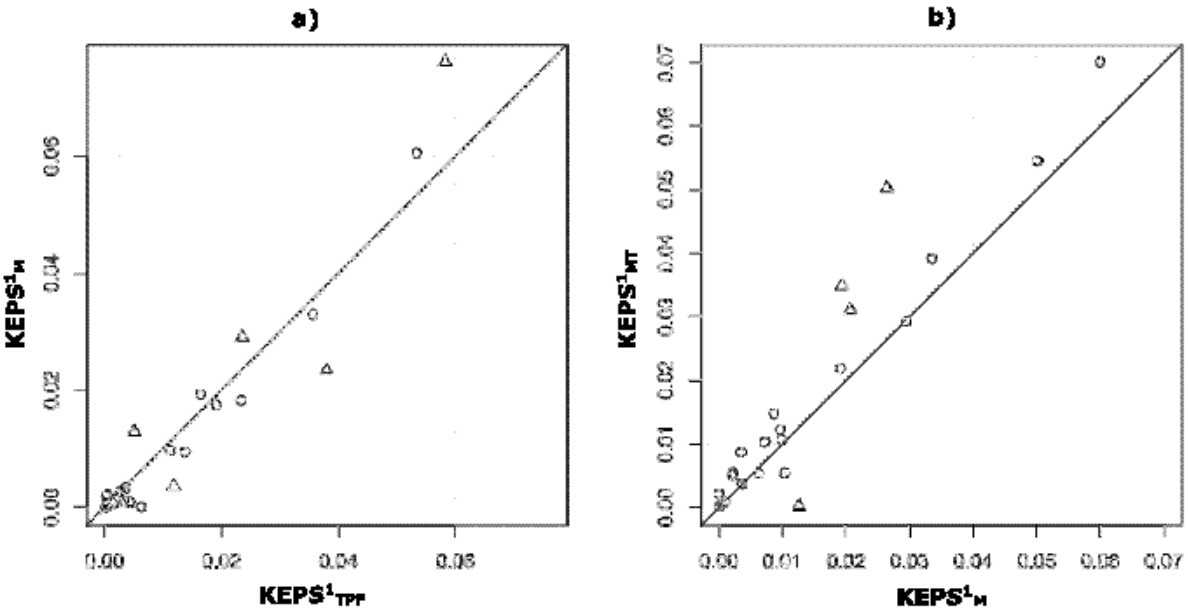


Figura 10.1: Estudio de la influencia de la regla de intercambio de soluciones

En estos diagramas cada punto representa una instancia de test y muestra la error relativo con respecto al óptimo para las dos metaheurísticas comparadas. Se define este error como $d = \frac{q\ q*}{q*}$. Cada punto es la media sobre el total de ejecuciones. En este tipo de diagramas, cuando un punto está por debajo de la diagonal significa que la estrategia del eje X tiene un peor valor medio que la estrategia del eje Y, y viceversa. Cuando un punto se representa por un triangulo indica que la diferencia es estadisticamente significativa (con un grado de confianza del 95 % usando el test de Wilcoxon) mientras que en el caso opuesto, el punto se muestra como un círculo.

La Figura 10.1 a) muestra el rendimiento de las metaheurísticas que utilizan parámetros fijos obtenidos del conocimiento experto. Las diferencias en términos de resultados entre KEPS^1_{Mem} y $\text{KEPS}^1_{CP\ TPF}$ son solamente significativas en 5 casos, de los cuales 3 son a favor de $\text{KEPS}^1_{CP\ TPF}$ y 2 a favor de KEPS^1_{Mem}. En el resto de casos los resultados son muy similares.

Cuando los parámetros se ajustan dependiendo de la instancia utilizando los árboles, Figura 10.1 b), parece que $\text{KEPS}^1_{CP\ T}$ obtiene resultados ligeramente mejores que $\text{KEPS}^1_{Mem\ T}$. $\text{KEPS}^1_{CP\ T}$ supera a $\text{KEPS}^1_{Mem\ T}$ en la mayoría de instancias, con tres casos en los que las diferencias son estadísticamente significativas, mientras que esta condición solo se cumple en una ocasión cuando la diferencia tiene el signo opuesto.

10.5.2. Influencia del método de configuración de los parámetros

En este apartado se estudia hasta que punto la metaheurística mejora sus resultados cuando los parámetros de la metaheurística se configuran en función de la instancia que se está resolviendo, teniendo en cuenta el escaso número de instancias de entrenamiento.

Los resultados se muestran en la Figura 10.2. Comenzaremos el análisis con KEPS^1_{Mem} y $\text{KEPS}^1_{Mem\ T}$.

Analizando los resultados mostrados en la Figura 10.2 a), se puede observar como los resultados se deterioran al añadir los árboles de selección de parámetros obteniendose peores resultados, con diferencias significativas en 6 casos. Si comparamos $\text{KEPS}^1_{CP\ TPF}$ y KEPS^1_{Mem} se puede observar en la Figura 10.2 b), que el rendimiento de ambas metaheurísticas es aproximadamente el mismo.

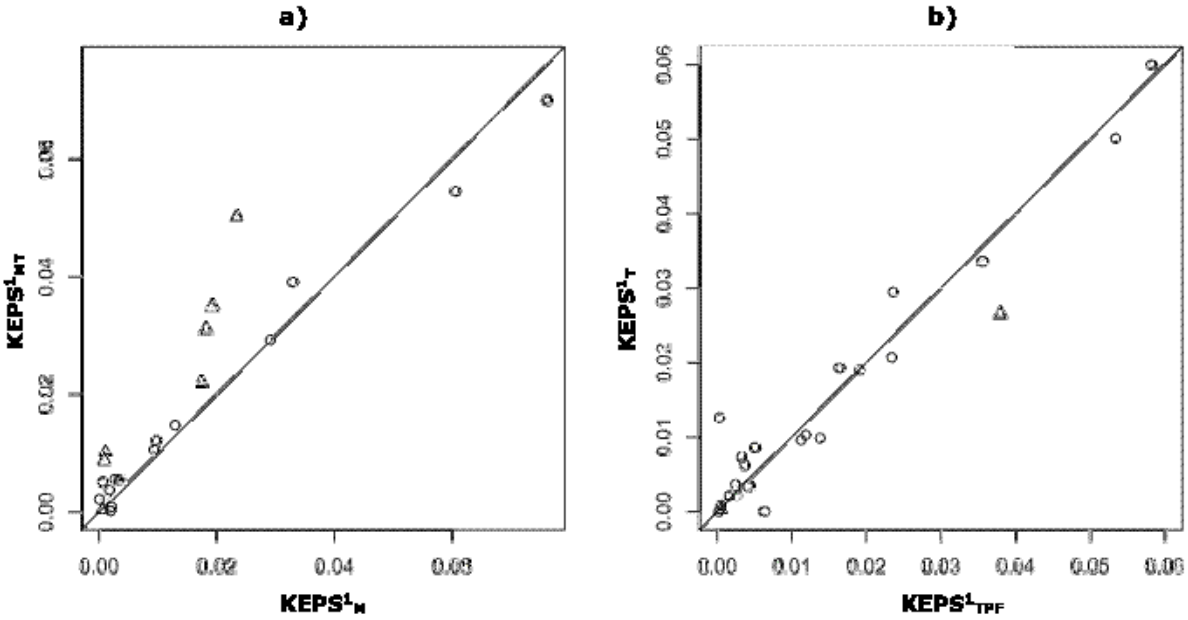


Figura 10.2: Estudio de la influencia de la selección de parámetros

Parte IV

NUEVOS ELEMENTOS EN EL DISEÑO DE METAHEURÍSTICAS COOPERATIVAS

Capítulo 11

Una evolución de la metaheurística cooperativa

En el Capítulo 5 se ha presentado y desarrollado un marco para el diseño de metaheurísticas cooperativas. Como se ha descrito, dicho marco contiene diversos elementos que sirven para ajustarla a diversos problemas. En el Capítulo 6 hemos realizado un estudio de esos elementos para comprobar el comportamiento de la metaheurística cooperativa cuando los variamos.

Pero debemos destacar que hay dos elementos que no hemos comprobado ni ajustado, y por tanto se desconoce el efecto que pueden tener sobre la metaheurística. Hablamos del "proceso de aprendizaje" de los modelos que definen la inteligencia del coordinador y de la "técnica" para construir dichos modelos, y por tanto, los tipos de modelos. En este capítulo, vamos a centrarnos en estos dos elementos, describiendo las modificaciones de ellos, y por consiguiente, definimos una evolución de las metaherísticas cooperativas tipo KEPS.

11.1. Mejorando la fase de preparado de datos

La metaheurística cooperativa que hemos descrito requiere de un tiempo necesario para llevar a cabo el aprendizaje. El objetivo de esta sección consiste en desarrollar un proceso que permita realizar la adquisición de conocimiento por parte del agente coordinador en un tiempo razonable. En concreto, el cuello de botella se encuentra en la fase de preparación de los datos ya que es necesario resolver diversas instancias con distintas metaheurísticas y distintos valores de sus parámetros para obtener las bases de datos de las que se extraerá el conocimiento, lo que puede ser costoso. El objetivo es construir un modelo que consiga un alto grado de precisión utilizando un subconjunto pequeño de las instancias disponibles.

Supuesto que el conjunto de datos de entrenamiento no contiene ruido, un modelo será tanto más preciso conforme más ejemplos haya adquirido para realizar el entrenamiento.

Resulta obvio sin embargo, que en un gran número de problemas no se puede realizar el aprendizaje de todo el conjunto de datos disponible. La principal razón es que el costo en cuanto a tiempo para realizar este proceso es prohibitivo. Otra razón importante es que incurriendo en este proceso de aprender todo el conjunto de datos disponible, se desaprovecha un amplio porcentaje de recursos aprendiendo información redundante. Esta situación justifica el planteamiento de estrategias para seleccionar el orden de aprendizaje de los ejemplos disponibles en el conjunto de entrenamiento, de modo que se dé prioridad a aquellas que aporten una mayor información al modelo.

11.1.1. Aprendizaje activo

Decimos que un agente está haciendo aprendizaje activo si en el proceso de aprendizaje tiene algún control sobre los ejemplos recibidos para el entrenamiento. Tal y como se destaca en [125], el teorema de Stone prueba que dado un conjunto lo suficientemente grande de ejemplos de entrenamiento, incluso los algoritmos simples, pueden comportarse de forma óptima. Sin embargo, existen bastantes situaciones en las que no podemos permitirnos obtener un gran número de ejemplos ya que es un proceso costoso. Además, la tarea de aprender tal conjunto de ejemplos puede requerir demasiados recursos en tiempo y memoria. Es por esto que en los últimos años ha habido un creciente interés en aplicar aprendizaje activo sobre una amplia diversidad de situaciones en las que resulta ventajoso elegir los ejemplos que parezcan más relevantes para ser incorporados al conocimiento del agente.

¿Cuándo interesa aplicar este tipo de aprendizaje? ¿Resulta siempre una mejor opción que el enfoque de aprendizaje pasivo? La clave está en el tiempo necesario para obtener la clasificacion (etiquetado) de un ejemplo. Si en el proceso de entrenamiento, el tiempo de clasificación de un ejemplo es despreciable, el hecho de emplear un enfoque activo frente al pasivo no nos ofrecerá ninguna ventaja. Sin embargo, en cuanto el tiempo de clasificación de un ejemplo es significativo, el aprendizaje activo comienza a tomar relevancia y consigue una precisión en la clasificación mucho mejor de lo que lo hace un enfoque pasivo. El aprendizaje activo también se puede utilizar para seleccionar de entre un conjunto aquellos que son significativos por estar cerca de la frontera de su clase, lo que puede permitir tener en un número reducido de ejemplos, aproximadamente la misma información que teníamos con el conjunto entero de ejemplos.

El aprendizaje activo trata de reducir el cuello de botella que supone el etiquetado de ejemplos en algunos problemas. Cuando el sistema de aprendizaje activo selecciona un ejemplo, pregunta a una parte u oráculo, cuál es la etiqueta de este ejemplo. El oráculo puede ser una persona, de modo que etiquetar nuevos ejemplos supone un costo importante, y además esta

persona no estará dispuesta a gastar su tiempo en clasificar muchos ejemplos. El oráculo también puede ser un programa el cual tiene un proceso costoso para obtener la etiqueta de un determinado ejemplo de entrada. En la Figura 11.1 se muestra porqué el aprendizaje activo es útil frente al aprendizaje tradicional o pasivo.

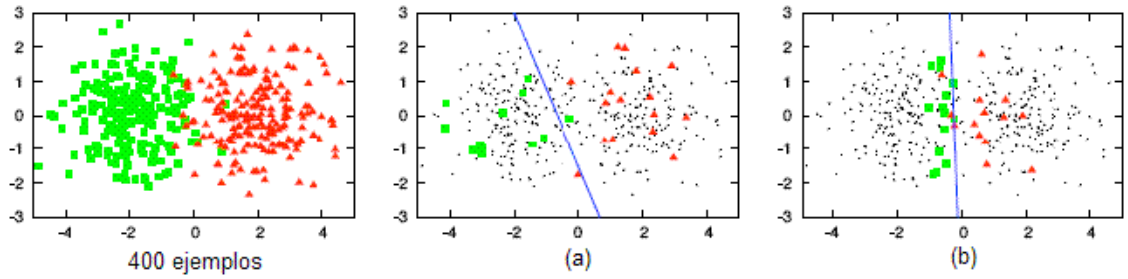


Figura 11.1: (a) Resultado del modelo tras aplicar aprendizaje pasivo seleccionando 30 ejemplos. (b) Resultado del modelo tras aplicar aprendizaje activo seleccionando 30 ejemplos.

Los ejemplos que se corresponden con una mayor mejora por parte del modelo que aprende son aquellos que comparten características entre dos o más clases, de modo que pertenecen a una clase y no a otra sólo por unas pocas razones. Algunos resultados demostraron que el número de ejemplos de entrenamiento se podían reducir sustancialmente manteniendo la precisión del modelo, si éstos eran seleccionados cuidadosamente. De esta forma podemos obtener un modelo que alcance la misma precisión que otro que utilice aprendizaje pasivo, pero en mucho menos tiempo tal y como refleja la Figura 11.2.

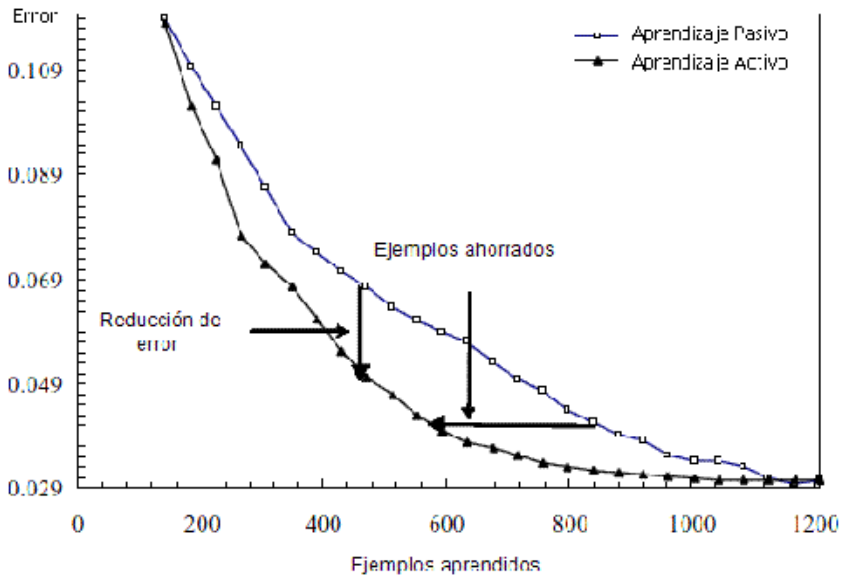


Figura 11.2: Modelos aprendidos mediante aprendizaje pasivo y activo, [236]

El esquema general de un algoritmo de aprendizaje activo se muestra en el Algoritmo 17.

Algoritmo 17: Esquema general de Aprendizaje Activo

Input:
L: Un conjunto de ejemplos etiquetados ;
UL: Un conjunto de ejemplos no etiquetados;
I: Una técnica para construir modelos;
Output:
Un modelo E inducido mediante I a partir del conjunto de ejemplos etiquetadas L;
begin
  while *no se cumpla el criterio de parada* **do**
    Aplicar I para construir un modelo con L como conjunto de entrenamiento;
    for $\{x_i \ / \ x_i \in UL\}$ **do**
      obtener su valor de información VI_i
    end
    Seleccionar un subconjunto S de tamaño M de UL basado en VI_i;
    Eliminar S de UL, etiquetar los ejemplos de S y finalmente añadirlos a L
  end
end

Se trata de un algoritmo en un nivel abstracto pues muchos de los elementos quedan parcialmente especificados. Estas cuestiones, así como ligeras modificaciones en este algoritmo, dan lugar a una amplia diversidad de técnicas propuestas para realizar aprendizaje activo sobre distintos tipos de problemas.

11.1.2. Tipos de enfoques de Aprendizaje Activo

Dentro del aprendizaje activo podemos encontrar varios tipos de enfoques en función de la capacidad que tiene el agente de elegir el nuevo ejemplo a inferir. A estos tipos se les llama escenarios y hay 3 principales: membership query synthesis, stream-based selective sampling y pool-based active learning (Figura 11.3).

Creación de las consultas (Membership Query Synthesis). En este tipo de escenario el agente puede pedir la etiqueta para cualquier ejemplo no etiquetado en el espacio de entrada incluyendo principalmente consultas de ejemplos que el agente puede crear o inventar en ese momento. Este tipo de escenario es a menudo tratable y eficiente para problemas de dominios finitos. Sin embargo en un buen número de situaciones resulta muy difícil crear nuevos ejemplos que sean válidos, especialmente si estos ejemplos arbitrarios deben ser etiquetados por un ser humano.

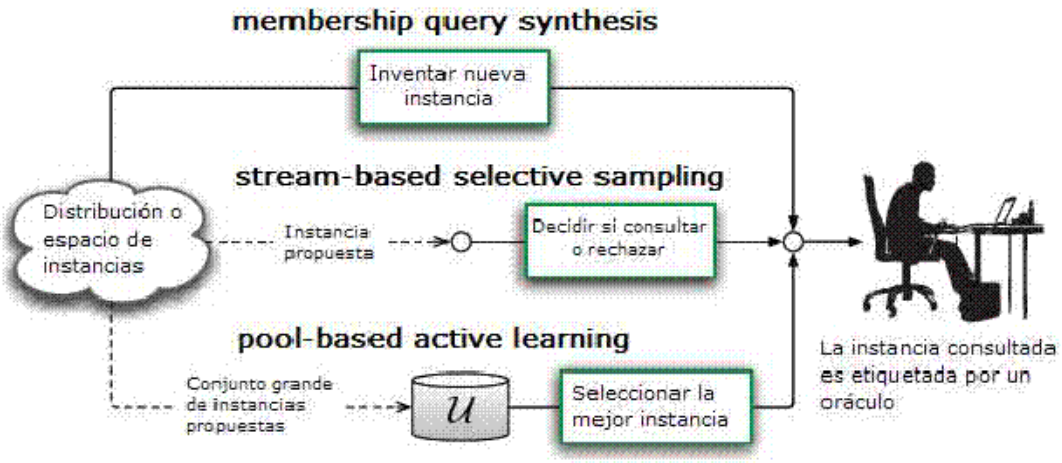


Figura 11.3: Escenarios de Aprendizaje Activo

Aprendizaje Activo Secuencial (Stream-Based Selective Sampling). Este escenario es útil cuando obtener un ejemplo sin etiquetar no presenta coste (o éste es despreciable) de modo que puede ser obtenido desde la distribución actual y entonces, el agente puede decidir si pedir al oráculo su etiqueta o no. En este escenario el agente recibe un ejemplo en cada iteración y debe decidir si le interesa o no.

Si la distribución de entrada es uniforme, el aprendizaje activo secuencial podría comportarse de forma parecida al aprendizaje membership query synthesis. Sin embargo, si la distribución no es uniforme o incluso es desconocida, el aprendizaje activo secuencial aún garantiza que las consultas respetarán la distribución subyacente.

Aprendizaje Activo Basado en Pool (Pool-Based Active Learning). Para muchos problemas reales, se podrían obtener de una vez grandes colecciones de ejemplos no etiquetados. Esto motiva este tipo de escenario, que asume que hay un pequeño conjunto de ejemplos etiquetados $\mathcal{L}$ y un gran conjunto disponible de ejemplos no etiquetados $\mathcal{U}$. Las consultas se seleccionan de este conjunto, el cuál se suele asumir cerrado, es decir estático, aunque esto no es estrictamente necesario. Normalmente, los ejemplos son elegidos de un modo voraz, de acuerdo a la medida de información usada para evaluar a todos los ejemplos del conjunto (o en caso de que $\mathcal{U}$ sea muy grande, un subconjunto).

La principal diferencia entre el aprendizaje activo secuencial y éste es que en el primero el agente recibe los ejemplos uno a uno, secuencialmente y toma decisiones individualmente mientras que el segundo evalúa y ordena la colección entera de ejemplos antes de elegir la mejor consulta.

11.1.3. Estrategias de consulta para el Aprendizaje Activo basado en Pool

Se han propuesto multitud de ideas para aplicar aprendizaje activo basado en pool a distintos problemas. A continuación se explican superficialmente las más utilizadas. Todas estas estrategias buscan obtener en cada iteración el ejemplo que contiene más información para el modelo. Se usará la notación x_A^* para hacer referencia al ejemplo más informativo según una estrategia determinada A.

Muestreo por incertidumbre (Uncertainty Sampling). En esta estrategia, el agente trata simplemente de pedir el ejemplo sobre el que tiene menos certidumbre. Es aplicable para ciertos modelos de aprendizaje probabilísticos que devuelven una probabilidad de pertenencia de un ejemplo a cada clase posible. Es decir, el modelo es capaz de realizar una estimación de $P(C|w)$ dado un ejemplo w y nos devuelve la probabilidad de que ese ejemplo pertenezca a la clase C. Así, si disponemos un modelo probabilístico para clasificar ejemplos entre dos clases C y D, se da que $P(C|w) = 1 \quad P(D|w)$. Por tanto, esta estrategia elegiría el ejemplo cuya probabilidad de pertenecer a una de las dos clases sea próxima a 0,5.

Cambio del Modelo Esperado. Esta estrategia trata de elegir el ejemplo que provocaría el mayor cambio del modelo actual si supiésemos su etiqueta. Debido a que los modelos probabilísticos son normalmente entrenados usando optimización basada en gradiente, el cambio del modelo puede ser medido por la longitud del gradiente de entrenamiento, es decir, el vector usado para reestimar los valores de los parámetros. Como el algoritmo no sabe de antemano la etiqueta correcta para un ejemplo dado, debe calcular el gradiente teniendo en cuenta todas las posibles etiquetas. Se ha demostrado en estudios empíricos que esta estrategia funciona bien pero puede ser computacionalmente costosa, sobre todo ante un conjunto grande de ejemplos de entrenamiento y un gran número de clases.

Reducción del error estimado. Esta técnica de aprendizaje activo se caracteriza por minimizar en cada consulta el error estimado resultante. Un proceso de aprendizaje activo óptimo es aquél que consulta aquel ejemplo de la muestra que, una vez incorporado al conjunto de entrenamiento, provoca que el error del modelo resultante sea el menor posible.

El problema es que para aplicar este paradigma sería necesario conocer de antemano la etiqueta y para cada ejemplo $< x, y >$ y en ese caso ya no tendría sentido el aprendizaje activo. En cambio se podría realizar una estimación del error que se obtendrá al añadir x al conjunto de entrenamiento y elegir aquel x cuyo error estimado sea menor. El principal problema de este modelo es su ineficiencia ya que no sólo requiere estimar el error futuro

para cada ejemplo elegido, sino que además el nuevo modelo debe ser reentrenado para cada posible consulta que a su vez itera sobre el conjunto total de ejemplos que tenemos en el pool.

Query By Committee. Query By Committee se basa en la idea de un conjunto de estudiantes que intentan aprender el concepto que posee el profesor (oráculo). Para conseguir dicho concepto, los estudiantes, antes de preguntar al profesor, deciden qué cuestión hacerle en función del desacuerdo que exista entre éstos en cuanto a la respuesta a esa pregunta. Cada estudiante tiene por tanto, un modo de percibir el concepto que trata de aprender, distinto (en general) al de los otros estudiantes. De esta forma habrá estudiantes que no coincidan en algún punto de dicho concepto. Interesará preguntarle al profesor aquellos aspectos del concepto que se trata de aprender donde existe un mayor desacuerdo entre los estudiantes. Una vez decidido qué aspecto van a preguntar al profesor, éste les contesta la solución, de modo que aquellos estudiantes que tenían una representación equivocada del concepto la corrigen y de forma colectiva se acercan más al concepto original que en principio tan sólo posee el profesor, de modo que disminuyen el espacio de búsqueda.

"Query By Committee" es una técnica simple pero muy potente y versátil, que se puede utilizar en un enorme rango de problemas de aprendizaje supervisado. Por ello, es la técnica que se ha escogido para integrar el aprendizaje activo en el aprendizaje de la configuración del agente coordinador.

11.2. Modificando el modelo para representar el conocimiento adquirido

En el contexto de aprendizaje activo, se han realizado varios proyectos utilizando un árbol como modelo de aprendizaje base, tales como [236]. Sin embargo el uso de este tipo de modelo no es muy común dentro del paradigma activo. De acuerdo a lo que se comenta en [106], las técnicas de construcción de árboles son inestables en el sentido de que se pueden construir árboles completamente distintos a partir de conjuntos de entrenamiento prácticamente iguales. Esto se debe al proceso recursivo en la construcción del árbol, que tiene en cuenta a todos los ejemplos de entrenamiento en cada momento. Esta inestabilidad resulta especialmente importante si se usan árboles como modelos para realizar aprendizaje activo. Esto es así ya que el algoritmo de aprendizaje activo adquiere un ejemplo (o un conjunto limitado de ellos) en cada iteración basándose en el modelo aprendido hasta ese momento. Sabemos que un nuevo ejemplo puede hacer cambios bruscos en los árboles construidos haciendo que varíe su precisión de predicción, ya sea para bien o para mal. Cuando se da este último caso, los

modelos del "comité" se basan en mala información para poder elegir el siguiente ejemplo.

Debido a la inestabilidad inherente a los árboles de decisión se ha decidido probar la utilización de otra técnica que se considera mejor adaptada al uso de aprendizaje activo de tal manera que se puedan comparar los resultados y utilizar aquella que sea más útil.

11.2.1. Máquinas de Vectores Soporte

Las máquinas de vectores soporte (SVM) son técnicas sencillas y de gran aplicación cuando se pretende construir un clasificador utilizando ejemplos. Un SVM está basado en la idea de minimización de riesgo estructural presentada en [249]. En muchas aplicaciones, los clasificadores SVM han demostrado tener un buen comportamiento [42]. Un SVM construye un modelo en forma de hiperplano n-dimensional el cual separa de forma óptima los ejemplos en dos clases. La estructura de un SVM es una red basada en kernels que realiza clasificación lineal sobre vectores transformados a un espacio de dimensión mayor. En este espacio, se construye un hiperplano que separa las clases de manera óptima (Figura 11.4).

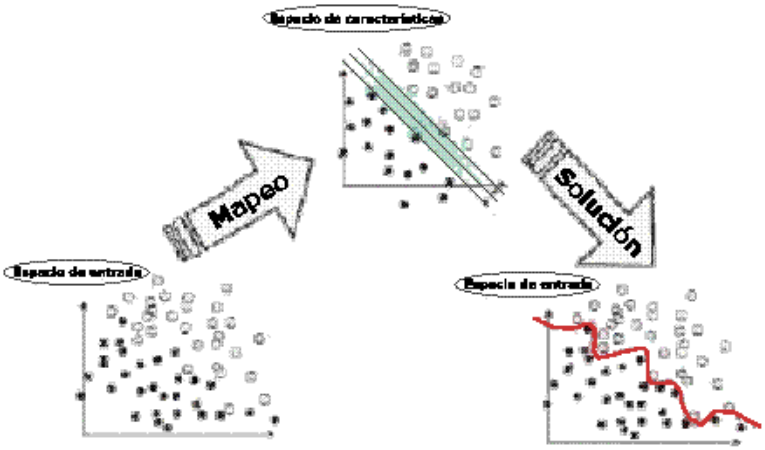


Figura 11.4: Proceso de un clasificador SMV

Los modelos de SVM están muy relacionados con las redes neuronales, de hecho, un modelo SVM que utilice una función kernel sigmoide es equivalente a una red neuronal de dos capas. En la nomenclatura utilizada en SVM, una variable predictora se llama atributo y un atributo transformado utilizado para definir el hiperplano se le llama característica. La tarea de escoger la representación más conveniente se conoce como selección de características. Un conjunto de características que describe un ejemplo (es decir, una fila de valores de predictores) se le llama vector. Así, el objetivo del modelado mediante SVM es encontrar el hiperplano óptimo que separa grupos de vectores de tal manera que ejemplos en una clase de la variable objetivo están en un lado del hiperplano y los ejemplos en otra clase están en el otro lado. Los vectores

cerca del hiperplano son los vectores soporte.

Supongamos un conjunto $S = \{(x_1, clase), \ldots, (x_l, clase)\}$ de l ejemplos de entrenamiento donde $clase = \{Clase_1, Clase_2\}$, como se observa en la Figura 11.5. Por tanto, un SVM primero mapea los ejemplos de entrada a un espacio de características de una dimensión mayor para encontrar un hiperplano que los separe y maximice el margen m entre las clases en este espacio como puede apreciarse en la Figura 11.6(a).

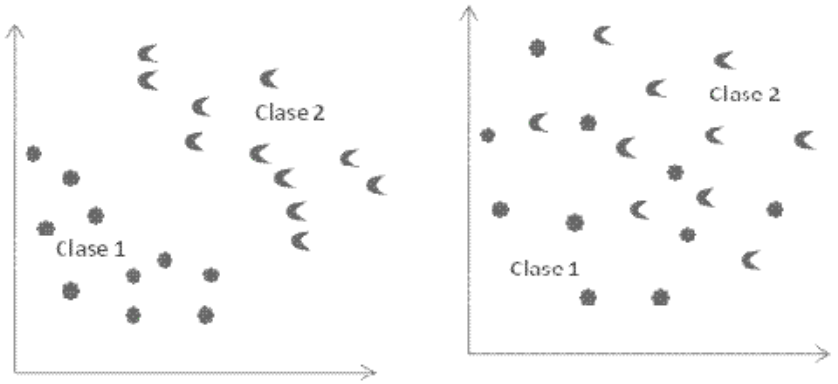
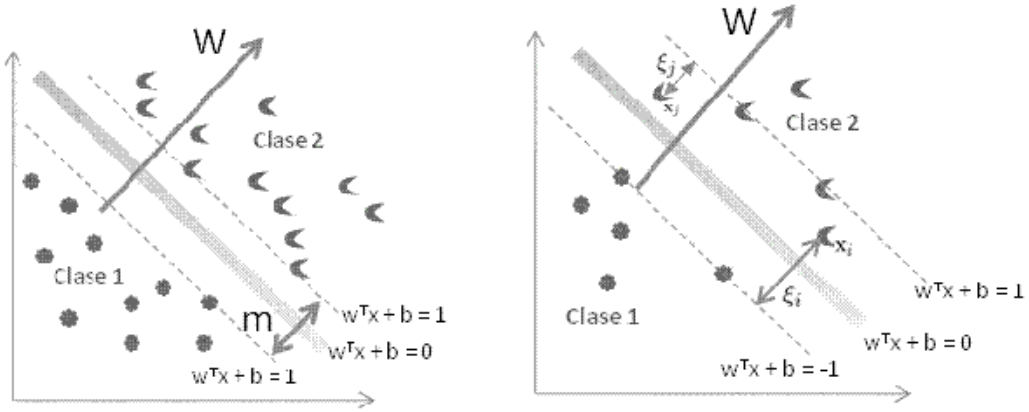


Figura 11.5: Conjunto de ejemplos de entrenamiento



(a) Frontera de decisión (b) Parámetro de error ξ_i

Figura 11.6: Espacio en características de un problema

Cada ejemplo de entrenamiento $x_i \in \mathbb{R}$ pertenece a alguna de las dos clases y se le ha asignado una etiqueta $y_i \in \{\ 1, 1\}$ para $i = 1, \ldots, l$, identificada por $Clase_1 = 1$ y $Clase_2 = \ 1$. Si lo que realizamos es una separación lineal de los datos, debemos encontrar el hiperplano $w \cdot x + b = 0$ definido por el par (w, b), tal que se pueda separar los ejemplos x_i

de acuerdo a la función

$$f(x_i) = w \cdot x_i + b = \begin{cases} \geq 1 & \text{si } y_i = 1 \\ \leq \quad 1 & \text{si } y_i = \quad 1 \end{cases} \tag{11.1}$$

donde $w \in \mathbb{R}^N$ y $b \in \mathbb{R}$.

Sea S cualquier conjunto de ejemplos (linealmente separables o no), este problema se puede resolver mediante el problema de programación cuadrática (11.2).

$$\begin{array}{ll} Min & \left\{ \frac{1}{2} w^2 + C \sum_{i=1}^{l} \xi_i \right\} \\ \text{s.a} & y_i (w \cdot x_i + b) \geq 1 \quad \xi_i \\ & \xi_i \geq 0, \ i = 1, \ldots l \end{array} \tag{11.2}$$

donde C es una constante (el parámetro C puede definirse como un parámetro de regularización). Los $\xi_i \neq 0$ en (11.2) son aquellos para los cuales el punto x_i no satisface (11.1) (véase la Figura 11.6(b)). El término $\sum_{i=1}^{l} \xi_i$ puede ser tomado como medida del error de clasificación (en el caso de $\xi_i = 0$ estamos en un caso de espacio de características linealmente separable).

En el caso de realizar una separación no lineal se utiliza una función $\phi(\cdot)$ que transforma el conjunto S de ejemplos a un espacio diferente, de modo que se realice una clasificación lineal sobre los ejemplos transformados. En esta situación, lo ejemplos aparecen como $\phi(x_i)$ y los pesos estarán en el nuevo espacio de dimensión mayor (Figura 11.7).

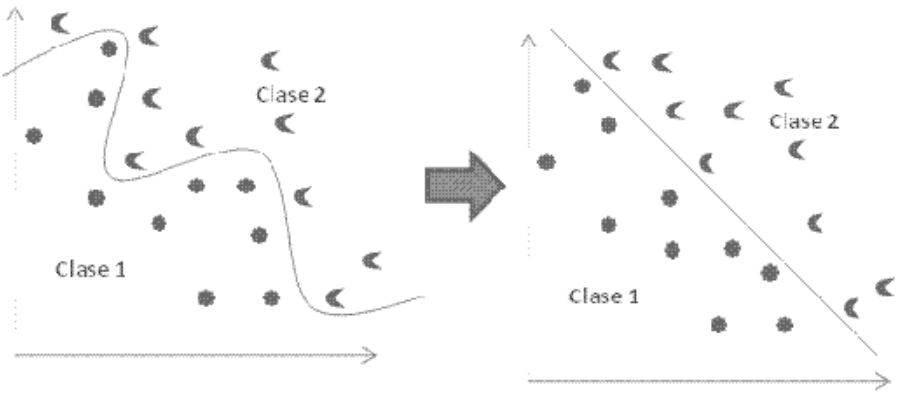


Figura 11.7: Transformación de los ejemplos

La formulación SVM pasa a ser el problema (11.3).

$$\begin{array}{ll} Min & \left\{ \frac{1}{2} w^2 + C \sum_{i=1}^{l} \xi_i \right\} \\ \text{s.a} & y_i (w \cdot \phi(x_i) + b) \geq 1 \quad \xi_i \\ & \xi_i \geq 0, \ i = 1, \ldots, l \end{array} \tag{11.3}$$

La transformación explícita suele ser costosa si $\phi(\cdot)$ es de alta dimensión. Por tanto y para resolver el problema sin tener que realizar dicha transformación explícita, se utiliza una función *kernel*, $K(\cdot,\cdot)$.

Aprendiendo un modelo mediante SVM. Buscar el hiperplano óptimo $(w \cdot x + b)$ en (11.2) o en el espacio transformado (11.3) es un problema de programación cuadrática (que recoge al caso linealmente separable), que puede ser resuelto construyendo un Lagrangiano y transformándolo en el problema dual (11.4) o (11.5), respectivamente.

$$\begin{array}{ll} Max & Q(\alpha) = \sum_{i=1}^{l} \alpha_i \quad \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} \alpha_i\alpha_j y_i y_j x_i \cdot x_j \\ \text{s.a} & \sum_{i=1}^{l} y_i\alpha_i = 0 \\ & 0 \leq \alpha_i \leq C, \quad i = 1,\ldots,l \end{array} \tag{11.4}$$

donde $\alpha = (\alpha_1,...,\alpha_l)$ es un vector de multiplicadores de Lagrange positivos asociados con las constantes de las restricciones de (11.2).

$$\begin{array}{ll} Max & Q(\alpha) = \sum_{i=1}^{l} \alpha_i \quad \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} \alpha_i\alpha_j y_i y_j \phi(x_i) \cdot \phi(x_j) \\ \text{s.a} & \sum_{i=1}^{l} y_i\alpha_i = 0 \\ & 0 \leq \alpha_i \leq C, \quad i = 1,\ldots,l \end{array} \tag{11.5}$$

donde $\alpha = (\alpha_1,...,\alpha_l)$ es un vector de multiplicadores de Lagrange positivos asociados con las constantes de las restricciones en (11.2) en el espacio transformado.

El algoritmo 18 muestra el proceso de aprendizaje de un modelo mediante SVM.

Algoritmo 18: Entrenamiento de SVM

begin
  $S = \{(x_1,y_1),\ldots,(x_l,y_l)\}$ conjunto de ejemplos donde $y_i \in \{\ \ 1,1\}$ para $i = 1,\ldots,l$ identificada por las dos clases de los ejemplos ($Clase_1 = 1$ y $Clase_2 = \ \ 1$);
  Calcular el hiperplano $\{x \ / \ w \cdot x + b = 0\}$ a partir los ejemplos de entrenamiento;
  if *no se ha definido una función kernel* $K(\cdot,\cdot)$ **then**
    Obtener α_i^* del problema (11.4);
    Calcular $w^* = \sum_{i=1}^{l}(\alpha_i^* y_i x_i) \quad y \quad b^* = 1 \quad w^*$
  else
    Obtener α_i^* del problema (11.4) en el espacio transformado;
    Calcular $w^* = \sum_{i=1}^{l}(\alpha_i^* y_i \phi(x_i)) \quad y \quad b^* = 1 \quad w^* \cdot \phi(x_s)$
  end
end

Clasificando mediante SVM. Una vez aprendido el hiperplano correspondiente que modela de manera óptima el conjunto de ejemplos de entrenamiento, lo utilizamos para realizar

la fase de inferencia de nuevos ejemplos de los cuales no disponemos el valor del atributo clase. El Algoritmo 19 muestra el proceso de inferencia mediante SVM.

Algoritmo 19: Inferencia mediante SVM

begin
  Sea x^* un ejemplo en el cual desconocemos la clase;
  if *las clases son linealmente separables* **then**
    $f(x) = \left(\sum_{i=1}^{l} \alpha_i y_i z_i \cdot z + b \right)$;
    if $f(x^*) > 0$ **then**
      la clase de x^* es $Clase_1$
    else
      la clase de x^* es $Clase_2$
    end
  else
    $f(x) = \left(\sum_{i=1}^{l} \alpha_i y_i K(x_i, x_j) + b \right)$ donde $K(\cdot, \cdot)$ la función kernel utilizada;
    if $f(x^*) > 0$ **then**
      la clase de x^* es $Clase_1$
    else
      la clase de x^* es $Clase_2$
    end
  end
end

SVM multiclase. Hasta ahora hemos supuesto que los ejemplos de que disponemos tienen dos posibles valores para el atributo clase ($Clase_1$, $Clase_2$). Para el caso de ejemplos que tienen un conjunto de posibles valores para la clase ($Clase_1$, $Clase_2$, $\cdots$, $Clase_n$), se propone el SVM multiclase. El enfoque para resolver este problema consiste en reducir el problema multiclase en un conjunto de problemas binarios. Los dos métodos más utilizados son:

- Uno contra todos. Se construyen todos los conjuntos posibles de dos grupos entre el conjunto de ejemplos. Cada conjunto estará formado por los ejemplos con clase $Clase_i$ y los ejemplos con las clases restantes ($Clase_1$, $Clase_2$, $\cdots$, $Clase_n$ menos la clase $Clase_i$).

- Uno contra uno. Se construyen todos los conjuntos posibles de dos grupos entre el conjunto de ejemplos. Cada conjunto estará formado por los ejemplos con clase $Clase_i$ y los ejemplos con la clase $Clase_j$.

A partir de estos conjuntos de problemas binarios construimos, mediante la fase de aprendizaje, todos los hiperplanos. La clasificación de nuevos ejemplos para el caso "uno contra todos" se realiza mediante el clasificador con la función de salida más alta (este clasificador es el que asigna la clase a los ejemplos a inferir). Para el caso "uno contra uno" se realiza mediante "voto". Cada clasificador asigna una clase al ejemplo a inferir, y la clase con mayor número de votos es la clase a asignar al ejemplo.

11.2.2. Uso de SVMs en estrategias de Aprendizaje Activo

Se han escrito una enorme cantidad de artículos en los que se utilizan los SVM como modelo base para aplicar aprendizaje con técnicas de aprendizaje activo [179,245,264]. Aplicar técnicas de aprendizaje activo es beneficioso en un modelo SVM y resulta adecuado usar SVM como modelo base para aplicar técnicas de aprendizaje activo.

Simbiosis entre SVMs y el Aprendizaje Activo. SVM sólo toma en consideración los ejemplos que están en la región de decisión (los cuales son denominados vectores soporte). Intuitivamente se observa cómo precisamente estos ejemplos tienen un alto grado de incertidumbre debido a que están en las fronteras de las clases. Éstos son los ejemplos que suelen ser consultados cuando se aplica una técnica de aprendizaje activo.

Aplicando aprendizaje activo sólo serán etiquetados los ejemplos con una alta probabilidad de ser usados como vectores soporte. De este modo, el clasificador podría aprender con un menor número de ejemplos una clasificación mejor, ya que en SVM sólo interesan los ejemplos que están en la frontera.

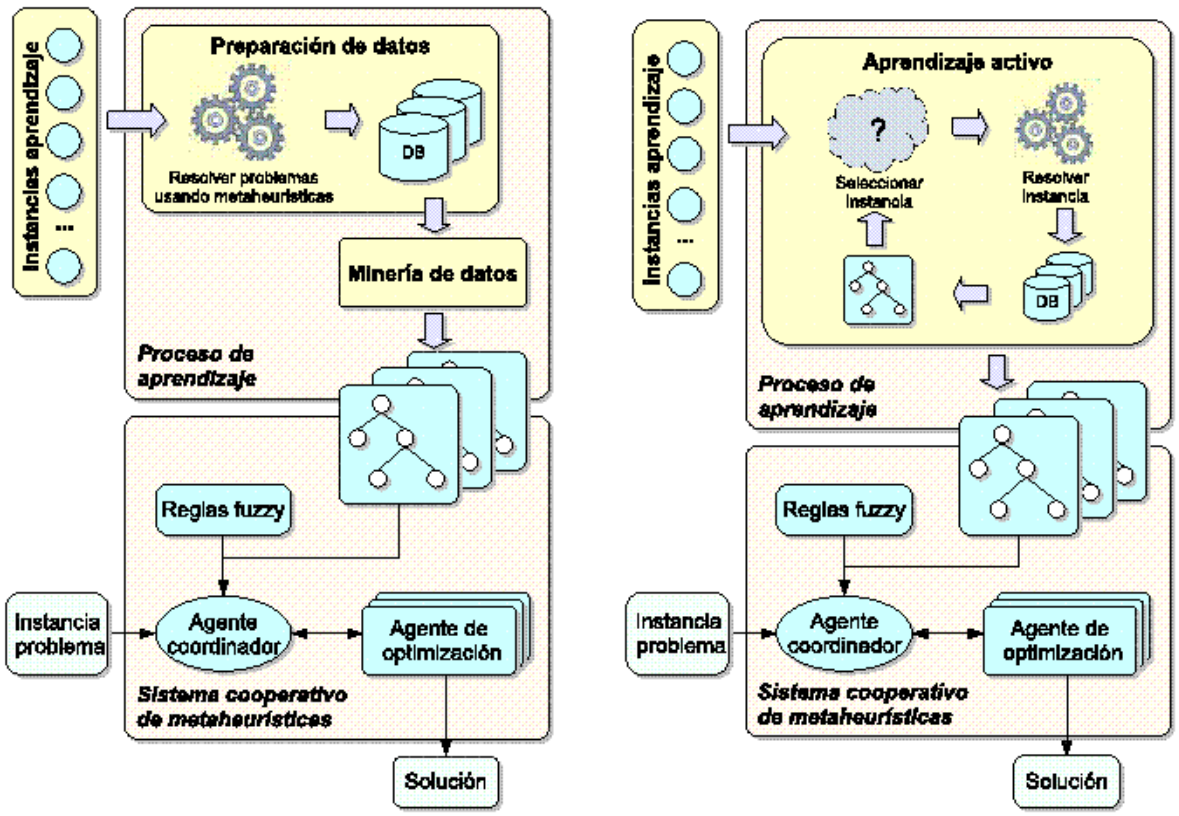
Por otro lado, las técnicas de aprendizaje activo seleccionan ejemplos que estén próximos a la frontera. Esto implica que el modelo que realiza el aprendizaje obtendrá una distribución de ejemplos distinta a la distribución de ejemplos inicial. Esto resulta importante tenerlo en cuenta en un gran número de algoritmos de aprendizaje donde el hecho de recibir una distribución similar al espacio de entrada se da por asumido. Las SVMs sin embargo, no son tan dependientes de esta cuestión como otros modelos, ya que éstas sólo toman en cuenta los ejemplos usados como vectores soporte.

Además, las SVMs utilizan los ejemplos de la frontera para encontrar el separador de mayor margen entre las clases. Al buscar el mejor separador, se realiza una búsqueda más efectiva de los ejemplos a etiquetar.

11.3. Una evolución de la metaheurística cooperativa

11.3.1. Aprendizaje activo en KEPS

Para poder introducir el aprendizaje activo en KEPS se debe sustituir el proceso de aprendizaje pasivo explicado en el Cap. 5, que mostramos en la Figura 11.8(a). El nuevo proceso lo mostramos en la Figura 11.8(b) en el cual los modelos son capaces de seleccionar inteligentemente cual debe ser la siguiente instancia que se debe incluir en los modelos de conocimiento.



(a) Pasivo (b) Activo

Figura 11.8: Proceso de Aprendizaje

Destacamos 2 situaciones diferentes:

- Aprendizaje de los pesos, ω_i, $i = 1, \ldots, n$, de las metaheurísticas: se debe entrenar el modelo para que infiera qué peso es adecuado para cada instancia y cada metaheurística.
- Aprendizaje de los parámetros de cada metaheurística: Cuando se recibe una instancia, es necesario inferir qué parámetros son los mejores para cada metaheurística. Por tanto,

existe un conjunto de problemas de clasificación en los que las etiquetas son los posibles valores que cada parámetro puede tomar. Es importante considerar que existen más de dos etiquetas.

En la primera situación se ha elegido utilizar una estrategia de comité. De este modo, cada iteración se calcula la varianza de la distribución obtenida cuando se infiere una instancia usando todos los miembros del comité. Por tanto se obtiene un vector de varianza de longitud n por cada instancia, siendo n el número de metaheurísticas consideradas. Finalmente se selecciona aquella instancia cuya suma de varianzas sea la mayor, siendo éste el modo de medir el desacuerdo entre los miembros del comité.

En la segunda situación se ha decidido utilizar otro comité para cada parámetro. Sin embargo, en este caso, se ha usado un criterio de selección distinto. Ahora no es necesario inferir los pesos, sino decidir de entre un grupo de etiquetas, por lo tanto estamos interesados en elegir la instancias cuya incertidumbre sea la mayor. [156] propone un método que ha reportado muy buenos resultados en problemas multiclase. Consiste en elegir la instancia que minimiza la diferencia entre la salida del comité para la clase más popular y la segunda más popular. Ésta es la opción escogida.

Se debe resaltar que en la primera situación no se puede utilizar la última opción, porque el problema no es de clasificación. Por ejemplo, si en la primera situación tenemos una instancia cuyo margen entre la clase más popular y la segunda más popular es bajo, esto no significa que esta clase tenga mucha incertidumbre, puesto que es posible que los valores estimados estén muy cerca de los pesos reales.

11.3.2. Probando la validez del aprendizaje activo en KEPS

El objetivo de esta sección es validar la estrategia propuesta de aprendizaje activo. Los criterios seguidos en los experimentos realizados están basados en los utilizados en publicaciones relacionadas con el aprendizaje activo tales como [236]. Existen casos donde hay ciertas modificaciones a estos criterios realizados en alguna de las pruebas. En estos casos se especificará qué cambios se han realizado y porqué.

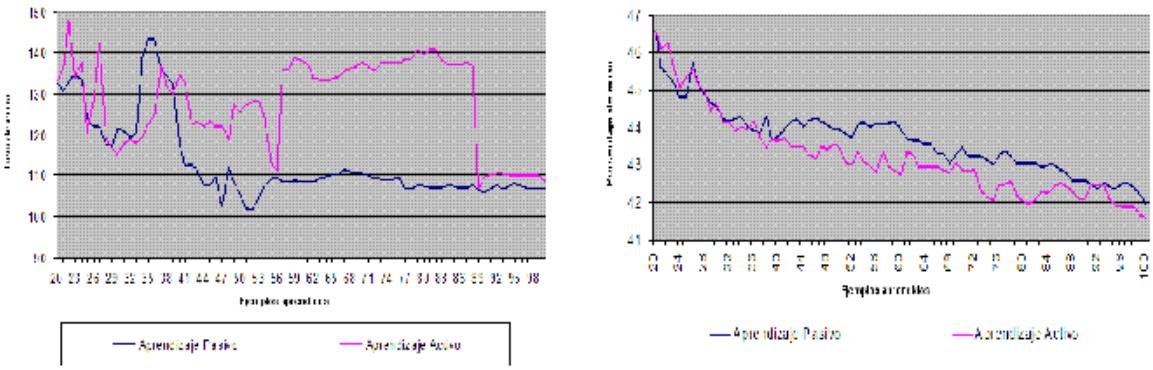
- Los parámetros principales del "comité" que se mantienen por defecto son los siguientes:
  - Número de ejemplos aprendidos previamente al inicio de la ejecución: 20.
  - Modo de elección de los ejemplos previos a la ejecución del comité: Elección de ejemplos al azar sin reemplazamiento.

- En la ejecución de distintos modos de aprendizaje, los ejemplos elegidos previamente son los mismos para cada modo con el objetivo de que el azar interfiera lo menos posible.

- Para las pruebas relacionadas con la metaheurística cooperativa se ha hecho lo siguiente:

  - En primer lugar se han resuelto un total de 2000 ejemplos, separados en dos grupos: uno de 1000 ejemplos para realizar las pruebas y otro con los restantes 1000 para el entrenamiento. Esto significa que el algoritmo de aprendizaje activo sólo decide sobre estos 1000 últimos.
  - Se ha repetido la ejecución de una misma configuración 20 veces de forma que se muestra la media resultante. En el caso de que existan pruebas donde estas medias estén cercanas entre sí o se pueda sospechar que la diferencia entre ambas no es significativa, se realizará un test estadistico de t de Student de diferencia de medias para poder decidir sobre esta cuestión.

Aprendizaje Activo respecto al Aprendizaje Pasivo. Vamos a validar el comportamiento del aprendizaje activo con los miembros del comité establecidos, contra el aprendizaje pasivo. El resultado se muestra en la Figura 11.9.



a) Predicción de pesos para las metaheurísticas

b) Predicción del valor de los parámetros

Figura 11.9: Comparación entre el enfoque pasivo y el activo

Analizando la gráfica de predicción de pesos (Figura 11.9a)), vemos que los resultados no son nada positivos, pues el modo de aprendizaje activo se comporta de un modo poco estable presentando en casi todo momento una precisión peor que el modo pasivo. Destaca especialmente el enorme salto hacia atrás que realiza el modelo activo a la altura del ejemplo 57. Tal y como se explicó, esto se debe a la inestabilidad que caracteriza a los algoritmos

de aprendizaje de tipo árbol de decisión, por lo cuál, al aprender un nuevo ejemplo puede cambiar toda la estructura interna dando lugar a importantes saltos en la precisión.

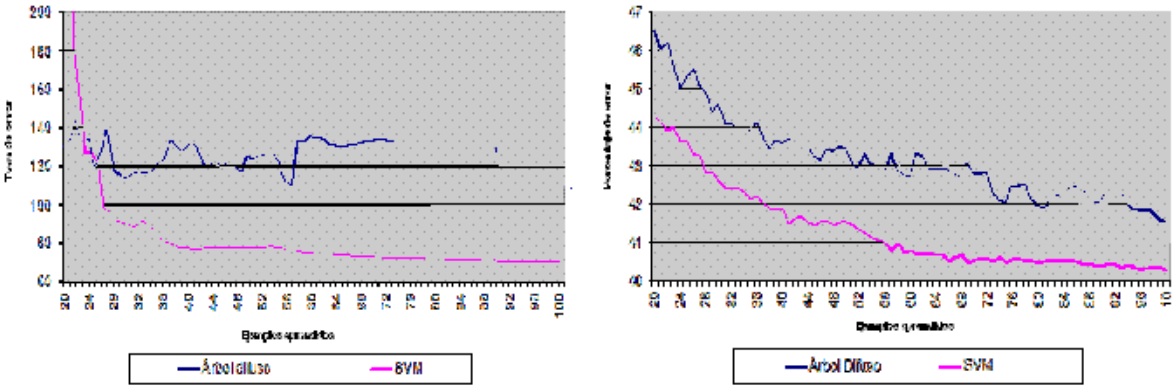
En la gráfica de los parámetros (Figura 11.9b)) podemos ver que la alternativa de aprendizaje activo sí que es capaz de superar al modo pasivo, manteniendo un porcentaje de error menor. En este caso se puede observar que, aunque existe un comportamiento poco estable en el modo activo, no resulta tan determinante como en el caso anterior de forma que mantiene una progresión adecuada manteniendo un mejor nivel que su homólogo pasivo.

11.3.3. SVM como modelo de aprendizaje en KEPS

Los resultados obtenidos muestran que el método de aprendizaje activo que se ha propuesto no consigue mejorar de manera remarcable los resultados del aprendizaje pasivo y en ocasiones son peores. Por esto se propone cambiar el modelo de aprendizaje, árboles de decisión, al modelo SVM ya que presenta mejores cualidades para ser usado en el paradigma de aprendizaje activo. El cambio sólo involucra al modelo. El algoritmo de aprendizaje activo basado en comité no se modifica.

11.3.4. Comparando FDT y SVM en el marco del aprendizaje activo

En la Figura 11.10 se muestra la comparación entre ambos modelos funcionando bajo el paradigma de aprendizaje activo.



a) Predicción de pesos para las metaheurísticas

b) Predicción del valor de los parámetros

Figura 11.10: Comparación usando aprendizaje activo con árboles y SVM

En la gráfica de los pesos (Figura 11.10a)), pese a que el modelo de SVMs empieza mal debido a los ejemplos iniciales elegidos al azar, pronto consigue un nivel de precisión muy bueno. Un aspecto determinante es el impacto que tienen los primeros ejemplos seleccionados

mediante aprendizaje activo en la precisión del modelo. En el caso de SVMs, los primeros ejemplos permiten obtener un modelo muy competitivo. Tanto es así, que el nivel alcanzado por el modelo de SVMs alcanza con 30 ejemplos una precisión tal, que el modelo basado en árbol difuso no es capaz de alcanzar ni en 100 ejemplos. Además, el comportamiento seguido por el modelo basado en SVMs parece más estable. Así, una vez superado el entusiasmo del modelo de SVMs con los primeros ejemplos, va consiguiendo mejorar poco a poco la precisión sin dar pasos hacia atrás.

En la gráfica de los parámetros (Figura 11.10b)) se observa como las SVMs superan claramente a los árboles ya desde el principio consiguiendo mantener la diferencia conforme se aprenden nuevos ejemplos. Además el aprendizaje con SVMs va aumentando la distancia horizontal respecto del aprendizaje con árboles conforme se van aprendiendo más ejemplos.

11.3.5. Comprobando la utilidad del aprendizaje activo

Comparando el aprendizaje activo y el pasivo. Antes de incluir el aprendizaje activo en el proceso de extracción del conocimiento es interesante probar su rendimiento desde un punto de vista clásico, es decir, usando los errores de clasificación y regresión. Las Figuras 11.11 y 11.12 comparan el rendimiento de las dos aproximaciones. El eje horizontal representa el número de instancias de entrenamiento y el eje vertical representa la tasa de error. La Fig. 11.11 muestra el rendimiento de las dos aproximaciones cuando se trata de predecir los pesos de cada metaheurística, y la Fig. 11.12 cuando se trata de predecir el valor del primer parámetro del algoritmo genético. El resto de figuras que mostrarían las comparaciones del resto de parámetros de las metaheurísticas se omiten por razones de espacio, pero se puede observar cómo la convergencia del aprendizaje activo es más rápida que la del aprendizaje pasivo y por tanto se puede esperar que el uso de aprendizaje activo puede ayudar a obtener una estrategia competitiva usando un número menor de instancias, que finalmente disminuirá el tiempo usado en el proceso de extracción del conocimiento.

Comparando los resultados obtenidos sin y con aprendizaje activo. Una vez se ha demostrado que el aprendizaje activo puede proporcionar buenos resultados, se pretende comprobar el rendimiento de la metaheurística usando los modelos obtenidos por el aprendizaje activo y el pasivo utilizando modelos SVM.

Para llevar a cabo los tests se ha utilizado un conjunto de 100 instancias del problema de la mochila-{0-1}, en el cual variaban tanto el tamaño como el modo en que se generaron las mismas. Con respecto al tamaño se utilizaron cuatro tamaños distintos: 500, 1000, 1500 y 2000 ítems. En cuanto al modo de generación se utilizaron los ya mencionados: Spanner

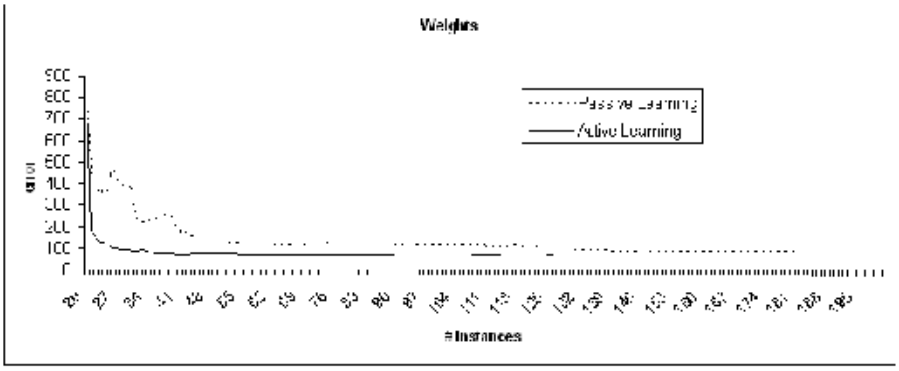


Figura 11.11: Curva de aprendizaje para la obtención de pesos

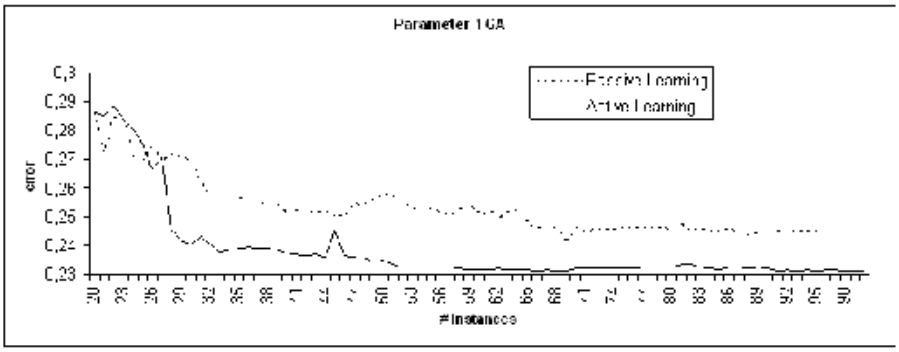


Figura 11.12: Curva de aprendizaje para la inferencia del primer parámetros del AG

uncorrelated (Unc span), Spanner weakly correlated (Wea span), Spanner strongly correlated (Str span), Profit ceiling (pceil) y circle.

Para realizar estos tests se ejecutaron dos metaheurísticas cooperativas utilizando cada una un paradigma de aprendizaje distinto que llamaremos $\text{KEPS}^1_{CP\ \ S_P}$ y $\text{KEPS}^1_{CP\ \ S_A}$ indicando respectivamente que el primero utiliza aprendizaje pasivo y el segundo activo. Cada instancia se resolvió 10 veces por espacio de 10 segundos y realizando la cooperación cada 250ms. Los resultados muestran la media de error y como subíndice la desviación típica.

Los modelos obtenidos con aprendizaje activo se construyeron usando 20 instancias, por otro lado los construidos usando aprendizaje pasivo utilizaron 500 instancias. La Tabla 11.1 muestra los resultados obtenidos por las dos aproximaciones, agrupados por tamaño y modo de generación. Como se puede observar, los resultados son muy similares, siendo los resultados obtenidos por el aprendizaje pasivo ligeramente mejores, sin que existan diferencias significativas. Sin embargo se debe notar el tiempo empleado en la generación de los modelos. Como se ha comentado con anterioridad, para clasificar una instancia se tiene que ejecutar varias veces con cada metaheurística usando distintos valores de parámetros. En este caso se ha

ejecutado cada instancia 5 segundos con 3 metaheurísticas diferentes, usando 8 combinaciones diferentes de valores de parámetros y repitiendo cada ejecución 10 veces. Esto significa que por cada instancia aprendida se necesitan 1200 segundos, o 30 minutos. Por tanto para 20 instancias son necesarios 400 minutos, aproximadamente 7 horas, y para 500 instancias casi 7 días. Este tiempo se puede reducir paralelizando el proceso de aprendizaje, sin embargo está claro que aunque el modelo que utiliza aprendizaje pasivo obtiene mejores resultados, los resultados obtenidos por el aprendizaje activo son también competitivos y la reducción del tiempo de aprendizaje los hace preferibles.

Tabla 11.1: Resultados de los tests

Type	$\mathbf{KEPS}^1_{CP-S_P}$	$\mathbf{KEPS}^1_{CP-S_A}$
unc span	$0{,}07_{0{,}20}$	$0{,}10_{0{,}25}$
wea span	$1{,}04_{1{,}07}$	$1{,}08_{1{,}11}$
str span	$1{,}59_{2{,}16}$	$1{,}68_{2{,}30}$
pceil	$0{,}35_{0{,}43}$	$0{,}35_{0{,}44}$
circle	$6{,}88_{4{,}39}$	$6{,}87_{4{,}34}$
Size		
500	$1{,}12_{0{,}87}$	$1{,}23_{1{,}00}$
1000	$1{,}69_{1{,}42}$	$1{,}63_{1{,}37}$
1500	$2{,}28_{1{,}94}$	$2{,}30_{1{,}97}$
2000	$2{,}85_{2{,}37}$	$2{,}90_{2{,}41}$

Comparando $\mathbf{KEPS}^1_{CP\ S_A}$ con $\mathbf{KEPS}^1_{CP\ T}$. Una vez se ha demostrado que la adaptación del aprendizaje activo para la metaheurística cooperativa funciona bien, pasamos a comparar la metaheurística cooperativa final con aquél que obtuvo los mejores resultados en el Capítulo 6, en el que se estudiaba la configuración de la metaheurística.

En el anterior apartado el control de las pruebas se realizó utilizando tiempo, tanto para gobernar el intercambio de soluciones como la condición de parada. Se siguió esta metodología debido a que se quería hacer hincapié en el tiempo ahorrado utilizando un paradigma de aprendizaje activo. Sin embargo en el Capítulo 6 las pruebas fueron realizadas usando evaluaciones, y en esta comparación se seguirá la misma aproximación.

Para llevar a cabo los tests se ha utilizado un conjunto de 20 instancias del problema de la mochila-{0-1}, en el cual variaban tanto el tamaño como el modo en que se generaron las mismas. Con respecto al tamaño se utilizaron cuatro tamaños distintos: 500, 1000, 1500 y 2000 ítems. En cuanto al modo de generación se utilizaron los ya mencionados: Spanner uncorrelated (Unc span), Spanner weakly correlated (Wea span), Spanner strongly correlated (Str span), Profit ceiling (pceil) y circle. Cada instancia se ejecutó 10 veces utilizando 100000 evaluaciones de la función objetivo. Los resultados muestran la media del porcentaje de error

y como subíndice la desviación típica.

La Tabla 11.2 y la Figura 11.13 muestran los resultados obtenidos por cada aproximación. Como puede observarse $\text{KEPS}^1_{CP\ S_A}$ obtiene mejores resultados que $\text{KEPS}^1_{CP\ T}$ en casi todas las instancias, siendo las diferencias significativas de acuerdo con el test de Wilcoxon con una confianza del 95 %. La Figura 11.13 corrobora estos resultados puesto que la mayoría de puntos se encuentran por debajo de la línea de separación y además representan diferencias significativas. Por tanto se puede observar como la utilización de SVM permite no solo reducir el tiempo de aprendizaje, sino también obtener mejores resultados que los que se pueden obtener con árboles fuzzy.

Tabla 11.2: Resultados de la comparación entre $\text{KEPS}^1_{CP\ T}$ y $\text{KEPS}^1_{CP\ S_A}$

tamaño	tipo	KEPS^1_{CP-T}	$\text{KEPS}^1_{CP-S_A}$
500	unc span	$4{,}58_{0{,}89}$	$2{,}56_{0{,}68}$
	wea span	$1{,}44_{0{,}24}$	$1{,}37_{0{,}22}$
	str span	$1{,}01_{0{,}04}$	$0{,}83_{0{,}06}$
	pceil	$0{,}11_{0{,}00}$	$0{,}11_{0{,}01}$
	circle	$7{,}62_{0{,}62}$	$6{,}05_{0{,}45}$
1000	unc span	$6{,}28_{0{,}96}$	$7{,}00_{0{,}66}$
	wea span	$2{,}29_{0{,}19}$	$2{,}12_{0{,}17}$
	str span	$1{,}13_{0{,}03}$	$0{,}99_{0{,}04}$
	pceil	$0{,}12_{0{,}00}$	$0{,}11_{0{,}00}$
	circle	$10{,}31_{0{,}56}$	$8{,}53_{0{,}28}$
1500	unc span	$8{,}51_{0{,}77}$	$9{,}49_{0{,}74}$
	wea span	$2{,}69_{0{,}15}$	$2{,}64_{0{,}09}$
	str span	$1{,}22_{0{,}03}$	$1{,}09_{0{,}03}$
	pceil	$0{,}12_{0{,}00}$	$0{,}12_{0{,}00}$
	circle	$10{,}60_{0{,}42}$	$9{,}39_{0{,}28}$
2000	unc span	$10{,}27_{0{,}60}$	$10{,}27_{0{,}62}$
	wea span	$3{,}02_{0{,}21}$	$2{,}99_{0{,}10}$
	str span	$1{,}25_{0{,}03}$	$1{,}15_{0{,}03}$
	pceil	$0{,}12_{0{,}00}$	$0{,}13_{0{,}00}$
	circle	$10{,}49_{0{,}40}$	$9{,}66_{0{,}14}$

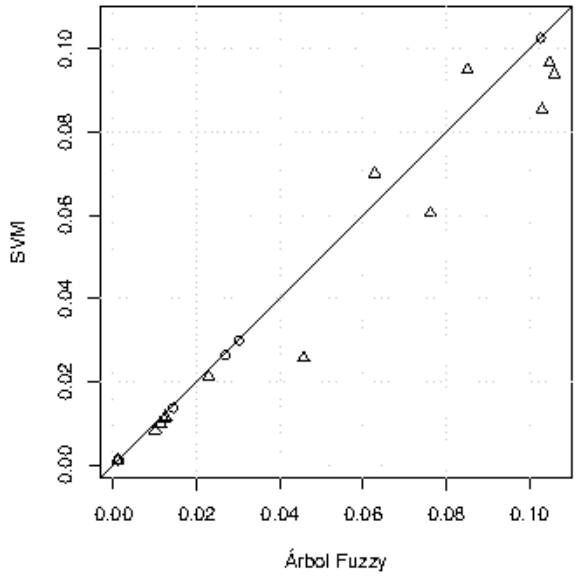


Figura 11.13: $\text{KEPS}^1_{CP\ \ T}$ vs. $\text{KEPS}^1_{CP\ \ S_A}$

Capítulo 12

Aplicando KEPS a problemas dinámicos

En el mundo real aparecen muchos problemas de optimización dinámicos, es decir, problemas en los que el espacio de búsqueda puede cambiar durante la ejecución. Para resolver este tipo de problemas es necesario encontrar estrategias que puedan rastrear el óptimo conforme se mueva en el espacio de búsqueda.

En este capítulo se utilizará la metaheurística cooperativa propuesta para resolver dos problemas de optimización dinámicos, con el objetivo de demostrar su aplicabilidad en estos entornos. En particular se resolverán el problema de la mochila {0-1} dinámico y el problema de los picos móviles. El primero de ellos es un problema de optimización combinatoria, mientras que el segundo es continuo, ambos son problemas muy populares ampliamente usados para probar nuevas estrategias en entornos de optimización dinámicos. La metaheurística cooperativa utilizada es $\text{KEPS}^1_{CP\ S_A}$ y será comparada con otras metaheurística tanto individuales como cooperativas para demostrar su competitividad.

12.1. Problemas de optimización dinámica

La investigación en el campo de las metaheurísticas se ha centrado tradicionalmente en los problemas de optimización estáticos, es decir, aquellos problemas donde el espacio de búsqueda permanece invariable durante la búsqueda. Sin embargo, los problemas de optimización reales raramente son estáticos, de hecho, suelen ser dinámicos, es decir, su espacio de búsqueda puede variar durante la ejecución.

Algunos ejemplos de este tipo de problemas son: el balanceo de la carga en redes de telecomunicaciones [238], donde las condiciones de tráfico varían a lo largo del tiempo, el control de un invernadero [248, 273] donde las condiciones ambientales pueden cambiar; el problema de alcanzar las fechas de vencimiento en un entorno de producción basado en

la planificación de tareas [240], con trabajos con diferentes pesos y tiempos de procesado inciertos, en el que pueden llegar trabajos nuevos; la optimización dinámica del modo de andar humano [11], donde el problema es calcular las historias de excitación de los músculos, las fuerzas de los músculos y los movimientos de los miembros sujetos al gasto mínimo de energía metabólica por unidad de distancia viajada.

Este tipo de problemas ha recibido el nombre de problemas de optimización dinámica (POD). Actualmente hay un interés creciente en este tipo de problemas y se están realizando grandes esfuerzos para adaptar las metaheurísticas clásicas a ellos.

Se han propuesto varios algoritmos y estrategias para resolver PODs, prevaleciendo aquellas estrategias que mantienen una colección de soluciones que evoluciona durante la ejecución, es decir, metaheurísticas basadas en poblaciones. La gran aceptación obtenida por estas metaheurísticas se debe a un idea simple: si se mantiene un población de soluciones entonces será más fácil localizar nuevos óptimos según cambie el espacio de búsqueda. De entre estas metaheurísticas las más utilizadas son los algoritmos evolutivos [38].

El principal problema que sufren los algoritmos evolutivos es la tendencia de la población a converger al mismo punto del espacio de búsqueda. Por esta razón se han hecho esfuerzos para mantener la diversidad de las poblaciones utilizando distintas técnicas como: la inserción de nuevos individuos generados de modo aleatorio [135], la utilización de nichos distintos [70], la modificación adaptativo de los parámetros [73], el uso de estructuras de memoria explícitas [37], etc.

Además de algoritmos evolutivos, se han utilizado otras metaheurísticas basadas en poblaciones que han obtenido resultados interesantes, tales como colonia de hormigas [12] o enjambre de partículas [28]. Otra línea de investigación interesante, que ha mostrado resultados excelentes, es el uso de metaheurísticas multipoblación, es decir, aquellas en que un conjunto de poblaciones evoluciona en paralelo para encontrar el óptimo. Estas metaheurísticas, aunque no sean ejecutadas en modo paralelo, realizan cooperación entre las distintas poblaciones. Los exploradores auto organizados, [39], son un claro ejemplo de estas aproximaciones, donde se propone un algoritmo evolutivo.

Pero no solo se han propuesto algoritmos evolutivos cooperativos, también se han popularizado las metaheurísticas basadas en enjambre de partículas de las cuales son claros ejemplos [29, 101]. Otra aproximación interesante es la propuesta en [213] donde se propone una estrategia multiagente, en la que diferentes agentes cooperan, implementando cada uno una estrategia de optimización muy simple.

El interés despertado por las metaheurísticas basadas en poblaciones, sin embargo, ha dificultado el desarrollo de metaheurísticas basadas en trayectorias (aquellas que exploran el

espacio de búsqueda realizando cambios iterativos en las soluciones iniciales). Sin embargo, trabajos recientes [131] los han señalado como metaheurísticas válidas capaces de obtener resultados similares a los obtenidos por metaheurísticas basadas en poblaciones.

Estos resultados sugieren que el uso de metaheurísticas cooperativas que incluyan tanto estrategias basadas en poblaciones como en trayectorias pueden ser usadas con éxito para tratar con los PODs, obteniendo resultados interesantes.

12.2. Metaheurísticas cooperativas en entornos dinámicos

Como se ha explicado en el Capítulo 3, las metaheurístcas cooperativas han sido aplicadas exitosamente a problemas estáticos. Sin embargo no se hizo mención a su rendimiento cuando se trata de resolver problemas dinámicos. En esta sección se comentarán diversas aproximaciones que tratan de resolver problemas de optimización dinámica mediante metaheurísticas cooperativas.

Debido a la popularidad alcanzada por los algoritmos evolutivos, no es de extrañar que hayan aparecido algunas metaheurísticas cooperativas, en las que varios de ellos cooperen, un claro ejemplo de ello son los self organizing scouts [39]. Pero no sólo se han propuesto metaheurísticas cooperativas basadas en algoritmos evolutivos, también dentro de las metaheurísticas basadas en poblaciones, la optimización de enjambre de partículas ha mostrado su validez y por tanto han aparecido también algunas implementaciones cooperativas como las propuestas en [29, 101].

Por otro lado, y más recientemente, han aparecido algunas metaheurísticas cooperativas basadas en metaheurísticas basadas en trayectorias. Como ejemplo podemos citar [213] en la que diferentes agentes cooperan paralelamente para resolver un POD, implementando cada agente una estrategia de optimización simple. También es interesante la metaheurística propuesta en [131] en la que diversas metaheurísticas basadas en trayectorias cooperan.

12.3. Adaptando KEPS a entornos dinámicos: KEPS+D

Hasta el momento, KEPS se ha presentado como un marco para el diseño de metaheurísticas capaces de adaptarse y obtener buenas soluciones para problemas de optimización estáticos. En este capítulo se aplicará a problemas de optimización dinámica para comprobar su robustez en este tipo de entornos. Sin embargo es importante realizar ligeros cambios en KEPS de tal manera que sea capaz de adaptarse mejor a los entornos dinámicos.

Para ello, el coordinador se encargará de monitorizar la instancia que se está resolviendo.

Tan pronto se ejecute un cambio, su configuración será revisada. Para hacer esto, el coordinador utilizará de nuevo los modelos SVM obtenidos del proceso de extracción inteligente del conocimiento, que son independientes de la instancia que se está resolviendo, y obtendrá nuevos pesos y valores de parámetros para todas las metaheurísticas.

Siguiendo la misma aproximación que otros investigadores en artículos recientes [246] se supondrá que el coordinador "sabe" cuando se produce un cambio en el espacio de búsqueda. Debemos admitir que esta suposición está lejos de ser una práctica ingenieril. Sin embargo, no hay dudas acerca de que la detección de cambios es todavía un reto tan difícil como el de adaptarse de un modo eficiente a los cambios. Por tanto, hemos decidido no resolver ambos problemas a la vez y centrarnos en uno de ellos, limitando la influencia del mecanismo de detección de cambios. Inmediatamente después del cambio todas las evaluaciones de las soluciones encontradas se vuelven inservibles, sin embargo, ninguno de los algoritmos empieza desde el principio. Estos no saben que tipo de cambio ha ocurrido, pero saben que es el momento de reevaluar las soluciones. Después de la reevaluación cada algoritmo continua el proceso de búsqueda.

Al nuevo marco que surge de adaptar KEPS a los problemas de optimización dinámicos se le ha dado el nombre de KEPS+D.

12.4. Análisis experimental

En esta sección se realizarán tests para comprobar la validez de la aplicación de KEPS+$D^1_{CP\ S_A}$ para resolver problemas de optimización dinámica. Para ello se utilizarán dos problemas, un problema de optimización combinatoria y un problema de optimización continua, que son respectivamente el problema de la mochila {0-1} dinámica (PMD) y el problem de los picos móviles (PPM). Ambos son problemas típicos de optimización dinámica y han sido ampliamente estudiados. Los resultados de KEPS+$D^1_{CP\ S_A}$ serán comparados con los obtenidos por las metaheurísticas que lo componen y con otras metaheurísticas cooperativas.

Los experimentos se llevarán a cabo en dos pasos. En el primero se comparan los resultados de KEPS+$D^1_{CP\ S_A}$ con los obtenidos por las metaheurísticas individuales que lo componen y con una metaheurística cooperativa simple (CS-WB). Esta metaheurística se obtiene usando el mismo sistema de metaheurísticas pero sustituyendo el coordinador por uno más simple, que sólo envía la mejor solución a la metaheurística que obtenga la peor solución.

Una vez se ha demostrado la efectividad de la metaheurística se realizará el segundo paso, en el que se compara KEPS+$D^1_{CP\ S_A}$ con una metaheurística cooperativa recientemente publicada [213].

12.4.1. Problemas

12.4.1.1. Problema de la Mochila {0-1} dinámica

El PMD es un problema NP-completo y uno de los problemas de optimización dinámica clásicos, que es además un excelente problema test para probar nuevos algoritmos en PODs [233]. Se puede definir formalmente como: dado un conjunto de objetos, cada uno con un coste y un beneficio asociados, determinar el subconjunto total que cumple que el coste total es menoer que un límite dado y el beneficio total es tan grande como sea posible. Su formalización matemática es:

$$
\begin{aligned}
Max \quad & \sum_{i=1}^{nob} p_i \times x_i \\
s.t. \quad & \\
& \sum_{i=1}^{nob} w_i \times x_i \ \leq \ C \\
& x_i \in \{0,1\} \, i = 1,\ldots,nob
\end{aligned}
$$

where

- nob es el número de objetos también conocido como tamaño de la instancia.
- x_i indica si el objeto i es incluido en la mochila.
- p_i es el beneficio asociado al objeto i.
- $w_i \in [0,\ldots,r]$ es el peso del objeto i.
- C es la capacidad de la mochila.

Se asume que $w_i < C, \ \forall i$ y $\sum_{i=1}^{nob} w_i > C$.

PMD fue introducido por primera vez por Goldberg y Smith en [130]. En este artículo se propuso una instancias del PMD muy simple en la que C cambiaba a lo largo del tiempo entre dos valores. Nosotros consideramos esta instancias demasiado simple para realizar nuestros tests. Afortunadamente, se han propuesto escenarios más interesantes:

- *Mori*: El primer tipo de instancias que se van a utilizar son las definidas en [197]. En estas instancias se tienen tres tamaños distintos 30, 60 y 90 objetos, con w_i y p_i elegidos aleatoriamente en el intervalo [1,500].

C cambia a lo largo del tiempo siguiendo el siguiente patrón: C se reduce del 90 % de c_{sum} al 10 % de c_{sum} con decrementos del 10 % de c_{sum} donde $c_{sum} = \sum_{i=1}^{nob} c_i$.

Se debe notar que las instancias definidas en [197] tienen sólo tamaño 30, pero se ha decidido generar instancias más grandes para añadir diversidad e incrementar su dificultad. De forma similar sus patrón de cambios solamente permite realizar 8 cambios, para permitir más cambios se ha decidido que una vez C sea igual al 10 % de c_{sum} vuelva al 90 %, siguiendo un patrón cíclico.

- *Karaman*: El segundo tipo de instancias fue definido en [163]. Dentro de este tipo se tienen cuatro tamaños distintos : 100, 200, 300 y 400 objetos. Cada objeto tiene un w_i y un p_i obtenido aleatoriamente en el intervalo [1000,5000] y C cambia aleatoriamente a lo largo del tiempo.

  Se debe notar que las instancias definidas en [163] tienen solamente tamaño 1000, pero se ha decidido generar instancias más grande para añadir diversidad e incrementar su dificultad.

- *Str span*: El último tipo de instancias fue propuesto recientemente en [233]. Estas instancias están inspiradas en por las instancias estáticas propuestas en [216] que han demostrado ser difíciles de resolver por algoritmos del estado del arte.

  En [233] se consideran solamente instancias spanner. Estas instancias son construidas a partir de un pequeño conjunto de parámetros conocido como conjunto spanner. Cada conjunto spanner está caracterizado por un par (v, m) donde v es el número de objetos en el conjunto y m un límite multiplicador. Se genera un conjunto de v objetos con pesos comprendidos en $[1, R]$ (donde R denota el máximo peso posible) y los pesos de acuerdo con alguna distribución. En [233] se utilizan las instancias fuertemente correlacionadas, tales que por cada $w_i \in [1, R]$, el beneficio correspondiente es $p_i = w_i + R/10$. Finalmente todos los objetos del conjunto spanner son normalizados. Los *nob* objetos de la instancia se construyen eligiendo aleatoriamente un objeto del conjunto spanner y un multiplicador aleatorio $a \in [1, m]$ tal que cada objeto es especificado por $(a \cdot w_i, a \cdot p_i)$. Por tanto los *nob* objetos se reparten en v conjuntos disjuntos, cada uno con un ratio beneficio-peso distinto.

  Para hacer el problema dinámico, el pso de un objeto spanner i denotado w_i, es una fracción del peso máximo, $\alpha_j R$, donde $\alpha_j \in [0, 1]$. Los pesos y por tanto el ratio beneficio-peso del grupo j pueden ser alterados cambiendo el valor de α_j. Las dinámicas están restringidas a estos coeficientes y cualquier tipo de función cuya salida (escalada) esté en el rango [0, 1] se puede usar para describir la trayectoria de cada α_j. En el caso que nos

atañe se utilizará la siguiente función: $\alpha_j(t+1) = \mu\alpha_j(t)(1 \quad \alpha_j(t)$ donde $\mu = 4$.

Para este escenario se han generado instancias con cuatro tamaños diferentes: 500, 1000, 1500 y 2000 objetos, el resto de parámetros toman los siguientes valores, $v = 2$, $m = 10$, $R = 10000$. Se debe notar que en [233] las instancias definidas tienen tamaño 20, pero se ha decidido generar instancias más grandes para añadir diversidad e incrementar su dificultad.

Finalmentes, se debe mencionar que para obtener el óptimo de cada instancia generada se usa el algoritmo exacto propuesto en [184].

12.4.1.2. Problema de los Picos Móviles

El PPM es uno de los PODs mas usados para probar la efectividad de estrategias propuestas [213]. PPM define un horizonte n-dimensional en el que se define un número prefijado de picos con localizaciones (X), alturas (H) y anchuras (W) especificas. Cada uno de los picos está distribuido aleatoriamente en un área restringida. Cada uno de los picos pude variar su altura, anchura y localización a lo largo del tiempo.

Con el objetivo de construir un puente entre problemas muy complejos y difíciles de comprender procedentes del mundo real y problemas de juguete muy simples, en [37, 198] se sugirió un problema con un horizonte multidimensional consistente en varios picos, en los que la altura, anchura y posición de cada pico se altera ligeramente cada vez que ocurre un cambio en el entorno.

Se debe notar que un pequeño cambio en el horizonte puede tener dos consecuencias: algunas veces puede ser suficiente adaptar la solución actual para alcanzar el nuevo óptimo, y en otras puede ser necesario cambiar a otro, que previamente fuera una solución ligeramente peor pero que ahora ha mejorado. La primera situación sucede cuando el óptimo se desplaza ligeramente, la segunda cuando la altura de los picos cambia de tal manera que un pico diferente se convierte en el pico máximo.

La función de test con n dimensiones y m picos se puede formular como:

$$F(\vec{x}, t) \quad = \quad \max(B(\vec{x}), \max_{i=1\ldots m} P(\vec{x}, h_i(t), w_i(t), \vec{p_i}(t)))$$

donde $B(\vec{x})$ es una "base" invariable en el tiempo, y P es la función que define la forma del pico, teniendo cada uno de los m picos su propios parámetros que cambian con el tiempo: altura (h), anchura (w), y posición ($\vec{p}$).

Las coordenadas, la altura y la anchura de cada pico se inicializan aleatoriamente. Entonces, cada Δe evaluaciones la altura y anchura de cada pico cambia añadiéndosele una variable gaussiana aleatoria. La posición de cada pico i se mueve usando un vector $\vec{v_i}$ de longitud fija s en una dirección aleatoria (para $\lambda = 0$) o una dirección dependiente de la dirección anterior (para $\lambda > 0$).

El parámetros s permite controlar la severidad del cambio, Δe determinará la frecuencia de cambio, λ permite controlar si los cambios siguen un patrón.

Formalmente un cambio de un solo pico se puede describir como:

$$\begin{aligned} \sigma &\in N(0,1) \\ h_i(t) &= h_i(t \quad 1) + height\_severity \cdot \sigma \\ w_i(t) &= w_i(t \quad 1) + width\_severity \cdot \sigma \\ \vec{p_i}(t) &= \vec{p_i}(t \quad 1) + \vec{v_i}(t) \end{aligned}$$

El vector de desplazamiento $\vec{v_i}$ es una combinación lineal de un vector aleatorio $\vec{r}$ y el vector de desplazamiento anterior $\vec{v_i}(t \quad 1)$, normalizado a longitud s, es decir:

$$\vec{v_i}(t) = \frac{s}{|\vec{r} + \vec{v_i}(t \quad 1)|}((1 \quad \lambda)\vec{r} + \lambda\vec{v_i}(t \quad 1))$$

El vector aleatorio $\vec{r}$ se crea tomando números aleatorios para cada dimensión y normalizando su longitud a s.

La función de complejidad se puede escalar fácilmente incrementando el número de dimensiones y/o el número de picos, o usando funciones complejas para los picos y la base.

Se han generado distintos entornos con PPM de acuerdo con ajustes estándar dados[1] :

- Escenario 1: horizonte de fitness definido por un espacio de búsqueda 5-dimensional con límites para cada dimensión entre $[0, 100]$. En este escenario existen cinco picos móviles que varian su altura en el intervalo $[30, 70]$, la anchura entre $[0.0001, 0.2]$ y la posición en una distancia de 1.
- Escenario 2: horizonte de fitness definido por un espacio de búsqueda 5-dimensional, con límites para cada dimensión entre $[0, 100]$. Sin embargo en el escenario 2 se tienen 5 configuraciones, que tienen 10, 20, 30, 40 y 50 picos que varían su altura en el intervalo $[30, 70]$, su anchura entre $[1, 12]$ y su posición en una distancia de 1.

[1] http://www.aifb.uni karlsruhe.de/~jbr/MovPeaks/

12.4.2. Configuración de la metaheurística cooperativa

Para definir completamente la metaheurística es necesario definir algunos de sus parámetros. El primero es qué metaheurísticas formarán parte de él: para los dos problemas se utilizarán las mismas, que son un Algoritmo Genético (AG), una Búsqueda Tabú (BT) y un Temple Simulado (TS). El pseudo-código de estas tres metaheurísticas se han presentado en los Algoritmos 12, 13 y 14, del Capítulo 6.

El siguiente aspecto que se debe discutir es el proceso de extracción del conocimiento seguido para diseñar los sistemas que tratarán cada uno de los problemas. Se debe notar que, aunque los problemas son dinámicos, el sistema tiene que aprender de problemas estáticos, que son aquellos que se pueden caracterizar. Esto no es un problema, puesto que los problemas dinámicos se pueden ver como una serie de problemas estáticos, además ya se ha comentado como el sistema es capaz de tratar los problemas dinámicos reconfigurandose después de cada cambio.

Un tema importante es el número de instancias que se usarán para realizar el proceso de extracción del conocimiento. Como se ha comentado antes, en el aprendizaje activo será necesario disponer de un conjunto inicial instancias, de gran tamaño, de las que se escogerá un conjunto reducido. Para los conjuntos iniciales se ha optado por tomar 25 instancias de cada tipo y tamaño. Por tanto los conjuntos iniciales para el PMD y PPM estarán compuestos por 275 y 150 instancias respectivamente. De todas estas instancias nos quedaremos con un 20 %, tomando como conjunto inicial el 10 %. Además cada instancia se resolvió 10 veces con cada combinación de parámetros, usando 4000 evaluaciones de la función objetivo y tomando distintas combinaciones de valores de parámetros para cada metaheurística.

12.4.3. Configuración de los experimentos

Los distintos escenarios disponibles se configuraron para que el problema cambiara después de 5000 evaluaciones de la función objetivo. Además en cada ejecución se realizaron 100 cambios. Para medir el rendimiento de cada estrategia se utiliza el error offline [38], una medida de rendimiento ampliamente aceptada para los PODs, que es la media de la diferencia entre el valor óptimo y el mejor encontrado en cualquier momento:

$$offline\ error = \frac{1}{T}\sum_{t=1}^{T}(optimum \quad bestSolution)$$

En cada subsección y una vez que se hayan mostrado los resultados, se realiza un análisis de los métodos usando técnicas estadísticas no paramétricas.

12.4.4. Resultados

12.4.4.1. Resultados para el PMD

En esta sección se prueba el rendimiento de la metaheurística cooperativa para el PMD. Para ello se compararán los resultados de KEPS+$D^1_{CP\ S_A}$ con los de las metaheurísticas individuales y CS-WB. Los resultados se muestran en la tabla 12.1.

Tabla 12.1: Results obtained solving DKP

type	size	AG	TS	BT	CS-WB	KEPS+$D^1_{CP-S_A}$
Mori	30	$1.61_{D.10}$	$2.87_{D.35}$	$3.33_{D.15}$	$0.75_{D.09}$	$0.42_{D.06}$
	60	$2.78_{D.09}$	$2.60_{D.16}$	$4.15_{D.20}$	$2.19_{D.14}$	$1.21_{D.07}$
	90	$3.96_{D.13}$	$5.77_{D.34}$	$3.52_{D.13}$	$2.79_{D.15}$	$1.81_{D.12}$
Karaman	100	$3.97_{D.11}$	$6.72_{D.22}$	$4.34_{D.18}$	$3.10_{D.20}$	$1.76_{D.11}$
	200	$7.42_{D.17}$	$11.01_{D.22}$	$4.69_{D.19}$	$4.11_{D.25}$	$2.72_{D.16}$
	300	$9.75_{D.16}$	$13.91_{D.20}$	$4.73_{D.19}$	$4.87_{D.31}$	$3.20_{D.18}$
	400	$11.71_{D.19}$	$18.11_{D.25}$	$4.78_{D.19}$	$5.00_{D.29}$	$3.18_{D.19}$
Str Span	500	$3.16_{D.07}$	$17.23_{D.68}$	$1.16_{D.71}$	$1.10_{D.14}$	$0.88_{D.10}$
	1000	$3.33_{D.05}$	$19.03_{D.46}$	$1.01_{D.29}$	$0.66_{D.10}$	$0.48_{D.09}$
	1500	$5.38_{D.06}$	$26.88_{D.69}$	$0.80_{D.35}$	$1.03_{D.10}$	$0.75_{D.08}$
	2000	$5.12_{D.06}$	$28.61_{D.41}$	$1.88_{D.55}$	$0.69_{D.08}$	$0.50_{D.05}$

De estos resultados se pueden extraer diversas conclusiones. Primero, si comparamos el rendimiento de las metaheurísticas individuales se puede observar como aquella que obtuvo mejores resultados es la búsqueda tabú, confirmado estadísticamente, seguida por el algoritmo genético, estadísticamente mejor que el temple simulado, y por último la que obtuvo peor rendimiento fue el temple simulado. Es interesant resaltar que los mejores resultados fueran obtenidos por la búsqueda tabú, que es una estrategia basada en trayectorias, las cuales se ha comentado con anterioridad que son descartadas generalmente cuando se resuelve un POD. Otra conclusión interesante es que todas las estrategias cooperativas obtienen mejores resultados que las individuales en prácticamente todas las instancias, además sus resultados son mejores estadísticamente, esto indica que la cooperación introducida es útil para la resolución de este problema. Finalmente si nos centramos en el rendimiento de las metaheurísticas cooperativas, se puede concluir que KEPS+$D^1_{CP\ S_A}$ es estadísticamente mejor que CS-WB, lo que indica que la inteligencia introducida en KEPS+$D^1_{CP\ S_A}$ es útil.

12.4.4.2. Resultados para PPM

Después de probar nuestras ideas con el PMD continuamos los experimentos con el PPM. Los resultados se muestran en la Tabla 12.2.

Tabla 12.2: Results obtained solving MPB

scenario	# of peaks	GA	SA	TS	CS-WB	$KEPS^1_{CP-S_A}$
scenario 1	5	$12.45_{1.74}$	$14.85_{1.28}$	$7.03_{0.63}$	$6.00_{0.77}$	$4.01_{1.11}$
scenario 2	10	$10.89_{1.98}$	$13.07_{1.11}$	$7.66_{0.32}$	$5.17_{0.30}$	$4.56_{0.31}$
	20	$10.08_{1.86}$	$8.04_{0.73}$	$5.42_{0.29}$	$3.57_{0.26}$	$2.96_{0.23}$
	30	$13.51_{2.02}$	$9.85_{0.84}$	$7.80_{0.33}$	$5.37_{0.32}$	$4.53_{0.36}$
	40	$12.61_{2.15}$	$7.05_{0.63}$	$5.71_{0.38}$	$3.86_{0.21}$	$3.40_{0.26}$
	50	$11.31_{1.54}$	$7.29_{0.68}$	$6.12_{0.24}$	$4.24_{0.20}$	$3.92_{0.26}$

Continuando con la estructura seguida en la anterior subsección empezaremos analizando los resultados de las metaheurísticas individuales. En este caso, una vez más la que obtuvo un mejor rendimiento es la búsqueda tabú, corroborado por los tests estadísticos, las otras dos metaheurísticas obtuvieron resultados estadísticamente equivalentes. De nuevo es importante notar que la metaheurística que mostró un mejor rendimiento es la búsqueda tabú, que no es una metaheurística basada en poblaciones. Si comparamos los resultados de las metaheurísticas individuales con los obtenidos por las metaheurísticas cooperativas, de nuevo, las cooperativas obtienen mejor comportamiento, corroborado estadísticamente. Por tanto una vez más combrobamos que la cooperación introducida ayuda de un modo efectivo a la resolución de los problemas dinámicos. De nuevo la metaheurística cooperativa que mostró un mejor rendimiento es KEPS+$D^1_{CP\ S_A}$, el uso de los modelos SVM para controlar la elección de los valores de los parámetros dependiendo del estado de la instancia combinado con la regla de intercambio de soluciones controlada por el rendimiento histórico de las metaheurísticas individuales demuestra que ayuda en estos entornos, demostrando su capacidad para adaptarse a entornos cambiantes.

12.4.4.3. Comparando KEPS+$D^1_{CP\ S_A}$ con *Agents* [213]

En esta sección se contrasta la validez de KEPS+$D^1_{CP\ S_A}$ comparándola con una metaheurística del estado del arte tal como Agents, propuesta en [213]. Esta metaheurística se basa en el uso conjunto de dos poblaciones. La primera compuesta por un conjunto de soluciones dispuestas en una parrilla M. Cada celda $M(i, j)$ contiene: (1) una solución del problema; y (2) su coste correspondiente. Las soluciones adyacentes en la parrilla no comparten ninguna relación, pudiendo representar puntos completamentes diferentes del espacio de búsqueda.

La segunda población contiene un conjunto de "agentes" $A = \{a_1, a_2, \ldots, a_n\}$, donde el número de agentes es mucho menor que el tamaño de M. Los agentes son capaces de moverse dentro de la parrilla y cambiar las soluciones almacenadas en ella. Usando distintos métodos

de cooperación y diversificación, son capaces de obtener resultados significativos.

Para realizar las comparaciones tenemos que llevar a cabo nuevos experimentos debido al hecho de que los resultados presentados en [213] se centran en instancias del PPM con 10 y 100 picos y con un espacio de 5 o 10 dimensiones.

La Tabla 12.3 muestra los resultados obtenidos por ambas metaheurísticas. Como se puede observar los resultados obtenidos por KEPS+$D^1_{CP\ S_A}$ son mejores que los obtenidos por Agents, de hecho la mejoría ha sido confirmada estadísticamente, y además su desviación típica es mucho menor. Estos resultados demuestran la validez de KEPS+$D^1_{CP\ S_A}$ para resolver PODs.

Tabla 12.3: Comparison with *Agents*.

# of dimensions	# of peaks	Agents	$KEPS^1_{CP-S_A}$
5	10	$5.84_{5.3}$	$4.56_{0.31}$
5	100	$5.66_{5.31}$	$4.17_{0.25}$
10	10	$11.25_{12.44}$	$7.39_{0.53}$
10	100	$9.79_{9.09}$	$7.60_{0.69}$

Capítulo 13

Conclusiones y Trabajos Futuros

En esta memoria se ha presentado un marco general para el diseño y construcción de metaheurísticas híbridas, paralelas, adaptativas y cooperativas basadas en sistemas multiagentes, el cual se ha denominado KEPS. En este marco, diversas metaheurísticas, tanto basadas en poblaciones, como basadas en trayectorias cooperan para encontrar la mejor solución posible a una instancia de un problema de optimización. En particular, una metaheurística tipo KEPS elige de manera inteligente, y dependiendo de la instancia a resolver, los parámetros de cada una de las metaheurísticas individuales que la componen. Estas últimas se ejecutan *paralelamente* mientras *cooperan* intercambiando sus soluciones de un modo *adaptativo*. La adaptatividad se consigue mediante el uso de un conjunto de reglas fuzzy capaces de evaluar la información extraída por un proceso de extracción de conocimiento preliminar. La propuesta ha sido aplicada a diversos problemas de optimización de varios tipos (problemas de optimización combinatoria y continua, problemas estáticos y dinámicos, problemas del mundo real y de laboratorio) obteniendo resultados satisfactorios que demuestran su robustez y adaptabilidad.

A continuación se desglosan los avances realizados con respecto a los objetivos que se plantearon al comienzo de la memoria (Capítulo 1):

1. *Profundizar en el conocimiento del campo de las estrategias cooperativas y la aplicación actual de técnicas de Soft Computing.* Se ha realizado un amplio estudio de las estrategias cooperativas centrado en aquellas que han alcanzado un cierto grado de adaptabilidad gracias al uso de técnicas de Soft Computing, como son las metaheurísticas cooperativas. Dentro de éstas, se ha observado la falta de estrategias que hicieran uso de conocimiento previo para adaptarse a los distintos problemas e instancias de ellos. En particular, resulta interesante la inclusión de dicho conocimiento previo haciendo uso de técnicas de Soft Computing. El control y los modelos de aprendizaje fuzzy son técnicas validas

para este propósito.

Con este objetivo, en la memoria se han estudiado las metaheurísticas cooperativas y el impacto que tiene sobre ellas la utilización de conocimiento previo añadido mediante técnicas de Soft Computing. En función de este estudio y algunas pruebas preliminares, se han observado mejoras al utilizar estrategias cooperativas que utilizan conocimiento previo.

2. *Diseñar y Especificar un protocolo de generación de datos para el estudio de la relación algoritmo-instancia (en nuestro caso, heurísticas - problemas).* La resolución de problemas utilizando metaheurísticas cooperativas que utilicen conocimiento previo ha demostrado ser eficiente, pero para obtener este conocimiento es necesario definir un proceso de extracción inteligente de conocimiento que permita obtener la configuración del coordinador a través de ejecuciones anteriores de las metaheurísticas. Este proceso se encarga, para cada instancia del problema, de: 1) averiguar qué valores de parámetros son los mejores para cada metaheurística y 2) obtener un ranking en base a pesos entre las metaheurísticas que cooperan. O lo que es lo mismo, realizar un estudio automático de la relación algoritmo-instancia. Con este objetivo, en el Capítulo 5 se define un proceso de extracción inteligente del conocimiento que ayuda a recoger relaciones algoritmo-instancia, y si bien no es capaz de resolver el problema de la selección del algoritmo, se apoya en la base del conocimiento previo para discernir cómo configurar una metaheurística cooperativa para que se adapte, de modo automático, a la instancia que se desee resolver. Dentro de este proceso de extracción, la primera fase definida se corresponde con el protocolo antes mencionado. En esta fase se reúne la información generada por la ejecución de distintas metaheurísticas y se estructura para su explotación. En particular, se define un protocolo dividido en dos partes, en la primera se ejecutan varias veces las metaheurísticas utilizando distintas combinaciones de parámetros y almacenando la información generada en conjuntos de datos llamados "crudos". En la segunda parte se refina la información contenida en estos conjuntos de datos, obteniendo los llamados conjuntos de datos "refinados". Los distintos pasos para diseñar el protocolo de generación de datos han sido publicados en los congresos [49, 52, 53, 61, 62], en el capítulo de libro [63] y en las revistas científicas [55] y [2] (sometido), asimismo, se ha propuesto una herramienta que proporciona soporte a esta fase, cuyos avances se han publicado en los congresos [45, 60] y en el libro [46].

3. *Comprobar computacionalmente el protocolo de generación de datos.* El protocolo se ha utilizado en diversos problemas, y de los resultados obtenidos se puede concluir que los datos que se generan contienen información útil acerca de la relación algoritmo-instancia.

Sin embargo, su aplicación conlleva la utilización de muchos recursos computacionales, lo que puede provocar una dilatación en los tiempos de aplicación del mismo. Por ello, en el Capítulo 11 se contempla la modificación del protocolo para reducir su consumo de recursos.

4. *Profundizar en el estudio de extracción de conocimiento (Data Mining) así como en las distintas formas de abordar las fases que conllevan estas técnicas, para aplicarlas en el conjunto heurísticas - problemas.* Como ya se ha comentado, para estudiar la relación heurística-problema, se ha definido un proceso de extracción del conocimiento. Para diseñar este proceso se realizó un estudio previo de las distintas fases del proceso de extracción del conocimiento, mostrado en el apéndice C. En dicho apéndice también se muestra una revisión de las técnicas de minería de datos que se han considerado más interesantes para el proceso que se definió. En base a este estudio, se decidió que una de las técnicas más apropiadas para nuestros objetivos son los árboles de decisión fuzzy, debido a su potencia y comprensibilidad. La idoneidad de esta elección ha quedado patente en los resultados obtenidos por la aplicación del marco general KEPS al resolver distintos problemas. Los distintos pasos seguidos para definir el proceso de extracción del conocimiento han sido publicados en los congresos [49, 52, 53, 61, 62], en el capítulo de libro [63] y en las revistas científicas [55] y [2] (sometido). Además para ayudar a la obtención de los modelos de árboles de decisión fuzzy se definió un proceso de fuzzificación de los datos sometido al congreso [50].

   Sin embargo, el problema de consumo de recursos generado por el protocolo de generación de datos nos llevó a definir un nuevo proceso de extracción de conocimiento que utilizara técnicas capaces de aliviar este problema. En particular, se observó que las técnicas de aprendizaje activo serían idóneas para reducir sensíblemente la utilización de recursos sin que los modelos generados se vieran afectados. Por ello, se definió un nuevo proceso basado en aprendizaje activo para obtener la relación heurística-problema. No obstante, la técnica de minería de datos escogida (los árboles de decisión fuzzy) demostró que no era capaz de adaptarse a los nuevos requerimientos del proceso. Basándonos en el estudio mostrado en el apéndice C se utilizó la técnica SVM como una nueva posible técnica, demostrándose su efectividad cuando se aplicó a las pruebas desarrolladas en los Capítulos 11 y 12. El nuevo proceso de extracción inteligente de conocimiento, utilizando aprendizaje activo, ha sido publicado en el congreso [4].

5. *Aplicar las distintas técnicas de Minería de Datos a la información generada por el conjunto heurísticas - problemas.* Los dos procesos de extracción de conocimiento y, por tanto, las distintas técnicas de minería de datos propuestas fueron aplicados a problemas

de diversa índole, obteniéndose resultados satisfactorios en todos los casos, y por tanto quedando demostrado que las técnicas escogidas son capaces de extraer conocimiento útil para la resolución de problemas de optimización.

6. *Extraer conocimiento útil tanto para la resolución de problemas de manera más eficiente como para realizar mejoras en las metaheurísticas cooperativas.* Como ya se ha comentado, el proceso de extracción del conocimiento ha sido aplicado a diversos problemas. En todos los casos, las metaheurísticas cooperativas que fueron configuradas utilizando el proceso de extracción, mejoraron sus resultados con respecto a las que no hicieron uso del mismo, por tanto, podemos afirmar que el conocimiento extraído es útil y que además ayuda en la ejecución de metaheurísticas cooperativas.

7. *Proponer un marco general para el diseño y construcción de metaheurísticas híbridas, paralelas, adaptativas y cooperativas basadas en sistemas multiagentes.* En este estadío, se desarrolla un marco general para el diseño de metaheurísticas cooperativas que engloba el diseño de una arquitectura multiagente en la que metaheurísticas de todo tipo son capaces de cooperar bajo la supervisión de un coordinador que, haciendo uso de conocimiento previo extraído mediante un proceso de extracción inteligente de conocimiento, es capaz de elegir de manera inteligente los parámetros de cada una de las metaheurísticas y controlar su intercambio de soluciones. El uso del conocimiento adquirido se ha realizado mediante un conjunto de reglas fuzzy que deciden cuándo y cómo intercambian información las distintas metaheurísticas. La definición de este marco de trabajo ha sido publicada en el artículo de revista científica [55].

   Una vez definido este marco general, se ha profundizado en el estudio del impacto de los distintos parámetros que influyen en su comportamiento. Así, en el Capítulo 6 se estudia la influencia de la composición de la metaheurística cooperativa, comparando diversas composiciones homogéneas (es decir, compuestas por una única metaheurística individual replicada) con una heterogénea (compuesta por diversas metaheurísticas individuales) y distintas metaheurísticas individuales. En estas comparaciones, se concluye que una metaheurística cooperativa homogénea siempre supera en rendimiento a su homóloga individual, y cómo la metaheurística heterogénea es la que mejores resultados obtiene. A continuación se estudia qué tipo de arquitectura resulta más adecuada si 1) una arquitectura bicapa, con dos fases secuenciales, una relativa a metaheurísticas basadas en poblaciones y otra relativa a metaheurísticas basadas en trayectorias, o 2) una arquitectura monocapa, en la que todas las metaheurísticas se ejecutan en paralelo. Los resultados demuestran que ambas arquitecturas son válidas, obteniendo resultados similares. Como siguiente paso, se comprobó qué esquema de cooperación produce mejo-

res resultados, si uno basado en la memoria de la ejecución o en conocimento adquirido previamente. Los resultados muestran una mejora sensible al utilizar la segunda aproximación, por lo que se concluye que la utilización de un proceso de aprendizaje previo ayuda a las metaheurísticas cooperativas. Por último, se comprueba la influencia del número de intercambios de soluciones realizados entre las metaheurísticas individuales. En los resultados se observa que un número demasiado alto puede llevar a malos resultados, mientras que pocos intercambios también influyen negativamente. Un término intermedio de intercambios obtiene resultados muy satisfactorios. De igual manera, en el Capítulo 7 se estudian los beneficios de la cooperación y del uso de conocimiento cuando el tiempo de ejecución es una de las variables que gobiernan la cooperación. Finalmente, en el Capítulo 11 se estudia de que modo puede afectar el modelo de minería de datos utilizado en el proceso de extracción inteligente de conocimiento. El estudio del impacto de los parámetros que influyen en el comportamiento de las metaheurísticas generadas siguiendo este marco de trabajo ha dado lugar a la publicación de los capítulos de libro [54, 58].

8. *Evaluar distintas metaheurísticas cooperativas dentro del marco general aplicadas a problemas tanto de laboratorio como reales.*

   Las distintas metaheurísticas cooperativas construidas se han aplicado a diversos problemas. Así en los Capítulos 6, 7, 8, 9, 10 y 12 se realizan pruebas con problemas combinatorios, mientras que en el Capítulo 12 se prueba con un problema continuo. Por otro lado, en los Capítulos 6, 7, 8, 9 y 10 se prueba con problemas estáticos mientras que en el Capítulo 12 se prueba con dos problemas dinámicos. Además, en los Capítulos 6, 7, 10 y 12 se estudiaron problemas de laboratorio, sin embargo, en los Capítulos 8 y 9 se estudiaron problemas del mundo real. Finalmente, también se trató un problema de localización en el Capítulo 10. Se debe resaltar que en todos los problemas probados se obtuvieron resultados notables. Las distintas aplicaciones del marco general propuesto han dado lugar a la publicación de las comunicaciones a congreso [6, 56, 57, 59], del capítulo de libro [64], y de los artículos de revista científica [2,3]

Trabajos Futuros

El trabajo realizado deja abiertas muchas vías para posibles desarrollos posteriores:

- En cuanto a configuración del sistema, véanse los capítulos 6 y 7, existen numerosas mejoras que son susceptibles de ser aplicadas al modelo diseñado. Sería interesante profundizar en la aportación que produce al sistema cada metaheurística para saber discernir qué nuevas metaheurísticas serían interesante añadirle. De igual manera resultaría muy interesante enriquecer el conjunto de reglas que controlan el coordinador añadiendo nuevas reglas que traten de aprovechar otro tipo de conocimiento, o que intenten controlar otra situaciones que pueden aparecer durante la búsqueda. Asímismo se ha demostrado que dos parámetros de gran importancia en el sistema son el α-corte y la frecuencia de intercambios, que además deben ser convenientemente ajustados, por ello resultaría de gran utilidad hacer que el propio sistema fuera capaz de adaptarse a cada instancia eligiendo los mejores valores posibles.

- Con respecto al proceso de extracción del conocimiento todavía sería interesante investigar cómo se podrían paliar los problemas aparecidos en el Capítulo 10, en el que se observó que el sistema encuentra dificultades a la hora de encontrar conocimiento en problemas que tengan un número de instancias muy reducido.

- De igual manera, y continuando con el proceso de extracción del conocimiento, también nos parece muy interesante abordar el diseño de un nuevo proceso de extracción del conocimiento que sea capaz de aprender mientras el sistema se ejecuta. De esta manera el sistema podrá ajustarse aún mejor puesto que su conocimiento será obtenido de la propia ejecución y no a priori.

- Con respecto a los problemas de optimización dinámica, como se comentó en el Capítulo 12, se obvió el problema de la detección de los cambios en el espacio de búsqueda. Por tanto resultaría interesante dotar al coordinador de la inteligencia necesaria para detectarlos sin ayuda.

- Por otro lado también cabe preguntarse si sería viable la descentralización del sistema, eliminando la figura del coordinador de tal manera que los propios agentes de optimización fueran capaces de controlar su intercambio de soluciones, o incluso aprender a discernir cuáles son los agentes que pueden aportarles más ventajas y cuáles aquellos de los que no se espera un comportamiento fiable.

APÉNDICES

Apéndice A

Soft Computing. Conjuntos y Sistemas de Inferencia Fuzzy

"La peor decisión es la indecisión." – Benjamin Franklin

A.1. Introducción

Desde que Zadeh con su trabajo seminal, [266], introdujo el concepto de conjunto fuzzy y consecuentemente se extendió al concepto de variables lingüísticas, [267–269], la popularidad y el uso de la lógica fuzzy han sido extraordinarios. En 1994, L.A. Zadeh, [270], dio la primera definición de "Soft Computing", y hasta entonces las referencias a los conceptos que actualmente ésta maneja, solían hacerse de manera aislada con indicación del empleo de metodologías fuzzy. Zadeh propuso la definición de Soft Computing, estableciéndola en los siguientes términos:

> "Básicamente, Soft Computing no es un cuerpo homogéneo de conceptos y técnicas. Mas bien es una mezcla de distintos métodos que de una forma u otra cooperan desde sus fundamentos. En este sentido, el principal objetivo de la Soft Computing es aprovechar la tolerancia que conllevan la imprecisión y la incertidumbre, para conseguir manejabilidad, robustez y soluciones de bajo costo. Los principales ingredientes de la Soft Computing son la Lógica Fuzzy, la Neurocomputación y el Razonamiento Probabilístico, incluyendo este último a los Algoritmos Genéticos, las Redes de Creencia, los Sistemas Caóticos y algunas partes de la Teoría de Aprendizaje. En esa asociación de Lógica Fuzzy, Neurocomputación y Razonamiento Probabilístico, la Lógica Fuzzy se ocupa principalmente de la imprecisión y el Razonamiento Aproximado; la Neurocomputación del aprendizaje, y el Razonamiento Probabilístico de la incertidumbre y la propagación de las creencias".

Queda claro que la Soft Computing no está definida precisamente, sino que en una primera aproximación se define por extensión, por medio de distintos conceptos y técnicas que intentan superar las dificultades que surgen en los problemas reales que se dan en un mundo que es impreciso, incierto y difícil de categorizar.

A partir de ella, se han realizado varios intentos de ajustarla. Verdegay, Yager y Bonissone han presentado en [250] una definición más precisa e ilustrativa de lo que es la Soft Computing en la actualidad, en los siguientes términos:

> "El punto de vista que aquí consideramos es otra forma de definir la Soft Computing, por medio de la cual se la considera como la antítesis de lo que podríamos llamar Hard Computing. Este punto es consistente con el presentado en [270, 271]. La Soft Computing puede, por tanto, verse como un conjunto de técnicas y métodos que permitan tratar las situaciones prácticas reales de la misma forma que suelen hacerlo los seres humanos, es decir, en base a inteligencia, sentido común, consideración de analogías, aproximaciones, etc. En este sentido Soft Computing es una familia de métodos de resolución de problemas cuyos primeros miembros serían el Razonamiento Aproximado y los Métodos de Aproximación Funcional y de Optimización, incluyendo los de búsqueda. En este sentido, la Soft Computing queda situada como la base teórica del área de los Sistemas Inteligentes.".

En esta tesis nos movemos en las dos grandes áreas del Soft Computing: el Razonamiento Aproximado y la Aproximación Funcional/Métodos de Optimización, en concreto, en la lógica y conjuntos fuzzy y los árboles de decisión.

En la memoria utilizaremos los elementos básicos de la lógica fuzzy, conjuntos fuzzy, variables lingüísticas y su modelización mediante números fuzzy trapezoidales, etc. Por ello, en este capítulo se describen los elementos principales de lógica fuzzy y el particionamiento fuzzy, dado que serán utilizados para el diseño y construcción del clasificador base.

A.2. Lógica y Conjuntos Fuzzy

La Lógica Fuzzy se plantea como alternativa a la lógica tradicional con el objetivo de introducir grados de incertidumbre en las sentencias que califica, [272]. Hay situaciones en las que la lógica tradicional funciona perfectamente, sin embargo, el inconveniente de esta lógica es que en la vida real no nos encontramos frecuentemente con criterios de clasificación nítidos. Normalmente, la información no puede ser evaluada cuantitativamente de forma precisa, pero puede que sí sea posible hacerlo cualitativamente, y en estos casos hemos de

hacer uso de un *enfoque lingüístico*. Por ejemplo, cuando intentamos cualificar algún fenómeno relacionado con percepciones humanas, a menudo usamos palabras o descripciones en lenguaje natural, en lugar de valores numéricos. Supongamos que dado un conjunto de personas, las intentamos agrupar según su altura. Las personas no son simplemente *altas* o *bajas* sino que la mayoría pertenecen a grupos de altura intermedia. La gente suele ser *más bien alta* o *de altura media*. Casi nunca las calificamos con rotundidad, porque el lenguaje que usamos nos permite introducir modificadores que añaden imprecisión: *un poco, mucho, algo, ...*

Como la lógica tradicional es bivaluada (sólo admite dos valores: o el elemento pertenece al conjunto o no pertenece), se ve incapacitada para agrupar según su altura al anterior conjunto de personas, puesto que su solución sería definir un umbral de pertenencia (por ejemplo, un valor que todo el mundo considera que, de ser alcanzado o superado, la persona en cuestión puede llamarse *alta*). Si dicho umbral es 1.80, todas las personas que midan 1.80 o más serán *altas*, mientras que el resto serán bajas. Según esta manera de pensar, alguien que mida 1.79 será tratado igual que otro que mida 1.50, ya que ambos han merecido el calificativo de personas *bajas*.

Si dispusiéramos de una herramienta para caracterizar las alturas de forma que las transiciones entre las que son altas y las que no lo son fueran suaves, estaríamos reproduciendo la realidad mucho más fielmente. En la realidad hay unos puntos de cruce donde las personas dejan de ser *altas* para ser consideradas *medianas*, de forma que el concepto de *alto* decrece linealmente con la altura. Asignando una función lineal para caracterizar el concepto *alto* en lugar de definir un sólo umbral de separación estamos dando mucha más información acerca de los elementos. Esta función, como veremos, se llamará función de pertenencia.

En este sentido, el uso de la Teoría de Conjuntos Fuzzy ha dado muy buenos resultados para el tratamiento de información de forma cualitativa, [267–269]. El *modelado lingüístico fuzzy* es una herramienta que permite representar aspectos cualitativos y que está basada en el concepto de *variables lingüísticas*, es decir, variables cuyos valores no son números, sino palabras o sentencias expresadas en lenguaje natural o artificial, [267–269]. Cada valor lingüístico se caracteriza por un valor sintáctico o *etiqueta* y un valor semántico o *significado*. La etiqueta es una palabra o sentencia perteneciente a un conjunto de términos lingüísticos y el significado es un subconjunto fuzzy en un universo de discurso.

Se ha demostrado que es una herramienta muy útil en numerosos problemas, como por ejemplo en *recuperación de información*, *evaluación de servicios*, *toma de decisiones*, *procesos de consenso*, etc, [13, 24, 25, 151, 152, 183].

A.2.1. Conceptos Básicos

Vamos a comenzar presentando una revisión de los conceptos básicos de la Teoría de Conjuntos Fuzzy.

El interés de la Teoría de Conjuntos Fuzzy se centra esencialmente en modelar aquellos problemas donde los enfoques clásicos de la Teoría de Conjuntos y la Teoría de la Probabilidad resultan insuficientes o no operativos. Por ello, generaliza la noción clásica de conjunto e introduce el concepto de *ambigüedad*, de manera que los conjuntos fuzzy nos proporcionan una nueva forma de representar la imprecisión e incertidumbre presentes en determinados problemas.

A.2.1.1. Conjuntos Fuzzy y Funciones de Pertenencia

La noción de conjunto refleja la tendencia a organizar, generalizar y clasificar el conocimiento sobre los objetos del mundo real. El encapsulamiento de los objetos es una colección cuyos miembros comparten una serie de características o propiedades que implican la noción de conjunto. Los conjuntos introducen una noción de dicotomía, que en esencia es una clasificación binaria: o se acepta o se rechaza la pertenencia de un objeto a una categoría determinada. Habitualmente la decisión de *aceptar* se denota como 1 y la de *rechazar* como 0. Esta decisión de aceptar o rechazar se expresa mediante una función característica, según las propiedades que posean los objetos del conjunto. Sea U un conjunto cuyos elementos denotaremos por x, y sea A un subconjunto de U. La pertenencia de un elemento x de U al conjunto A viene dada por la función característica

$$\mu_A(x) = \begin{cases} 1 & \text{si y sólo si } x \in A \\ 0 & \text{si y sólo si } x \notin A \end{cases}$$

$\{0,1\}$ es el llamado conjunto valoración.

La Lógica Fuzzy se fundamenta en el concepto de *conjunto fuzzy*, [266], que suaviza el requerimiento anterior y admite valores intermedios en la función característica, que se denomina *función de pertenencia*. Esto permite una interpretación más realista de la información, puesto que la mayoría de las categorías que describen los objetos del mundo real, no tienen unos límites claros y bien definidos.

Un conjunto fuzzy puede definirse como una colección de objetos con valores de pertenencia entre 0 (exclusión total) y 1 (pertenencia total). Los valores de pertenencia expresan los grados con los que cada objeto es compatible con las propiedades o características distintivas de la colección. Formalmente podemos definir un conjunto fuzzy como sigue.

Definición A.1. Un *conjunto fuzzy* $\widetilde{A}$ sobre un dominio o universo de discurso U está caracterizado por una función de pertenencia que asocia a cada elemento del conjunto el grado con que pertenece a dicho conjunto, asignándole un valor en el intervalo [0,1], $\mu_{\widetilde{A}} : U \rightarrow [0,1]$ □

Así, un conjunto fuzzy $\widetilde{A}$ sobre U puede representarse como un conjunto de pares ordenados de un elemento perteneciente a U y su grado de pertenencia, $\widetilde{A} = \{(x, \mu_{\widetilde{A}}(x)) \ / \ x \in U, \mu_{\widetilde{A}}(x) \in [0,1]\}$. Por ejemplo, consideremos el concepto *persona alta*, en un contexto donde la estatura oscila entre 1 y 2 m. Como es de suponer, alguien que mida 1,30 m. no se puede considerar como *persona alta* por lo que su grado de pertenencia al conjunto de personas altas será de 0. Por el contrario, una persona que mida 1,90 m. sí la consideramos alta por lo que su grado de pertenencia al conjunto es de 1.

Las gráficas que representan una función de pertenencia pueden adoptar cualquier forma, cumpliendo propiedades específicas, pero es el contexto de la aplicación lo que determina la representación más adecuada en cada caso. Puesto que las valoraciones lingüísticas dadas por los usuarios son únicamente aproximaciones, algunos autores consideran que las funciones de pertenencia trapezoidales lineales son suficientemente buenas para capturar la imprecisión de tales valoraciones lingüísticas. La representación paramétrica es obtenida a partir de una 4-tupla (r, a, b, R), donde a y b indican el intervalo en que el valor de pertenencia es 1, con r y R indicando los límites izquierdo y derecho del dominio de definición de la función de pertenencia trapezoidal. Un caso particular de este tipo de representación son las valoraciones lingüísticas cuyas funciones de pertenencia son triangulares, es decir, $a = b$, por lo que se representan por medio de una 3-tupla (r, a, R). La figura A.1 muestra la descripción y la representación gráfica de un ejemplo de función de pertenencia trapezoidal.

$$\mu_{\widetilde{A}}(x, r, a, b, R) = \begin{cases} 0 & x < r \\ \frac{x - r}{a - r} & r \leq x \leq a \\ 1 & a \leq x \leq b \\ \frac{R - x}{R - b} & b \leq x \leq R \\ 0 & x > R \end{cases}$$

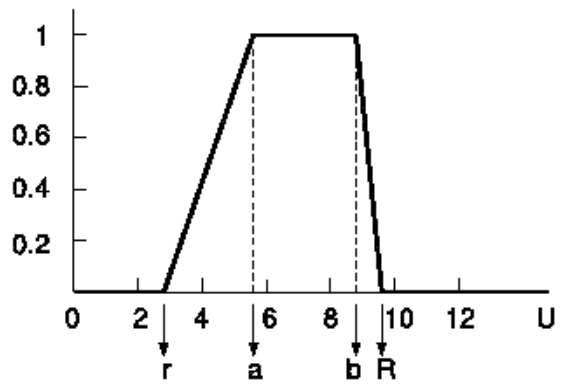


Figura A.1: Ejemplo de función de pertenencia.

A.2.1.2. Definiciones Básicas

Definición A.2. Dos conjuntos fuzzy $\widetilde{A}$ y $\widetilde{B}$ se consideran iguales ($\widetilde{A} = \widetilde{B}$) si y sólo si

$$\forall x \in X,\ \mu_{\widetilde{A}}(x) = \mu_{\widetilde{B}}(x)$$

□

Definición A.3. Se define *soporte* de un conjunto fuzzy $\widetilde{A}$ en el universo U, como el conjunto formado por todos los elementos de U cuyo grado de pertenencia a $\widetilde{A}$ sea mayor que 0:

$$supp(\widetilde{A}) = \{x \in U \ / \ \mu_{\widetilde{A}}(x) > 0\}$$

□

Definición A.4. La *altura* de un conjunto fuzzy $\widetilde{A}$ se define como el mayor grado de pertenencia de todos los elementos de dicho conjunto:

$$h(\widetilde{A}) = max\{\mu_{\widetilde{A}}(x) \ / \ x \in U\}$$

□

Definición A.5. El *α-corte* de un conjunto fuzzy $\widetilde{A}$ es el conjunto formado por todos los elementos del universo U cuyos grados de pertenencia en $\widetilde{A}$ son mayores o iguales que el valor de corte $\alpha \in [0,1]$:

$$A_\alpha = \{x \in U \ / \ \mu_{\widetilde{A}}(x) \geq \alpha\}$$

□

Los conjuntos A_α, $\alpha \in [0,1]$ constituyen una sucesión decreciente. Si $\alpha_1 \geq \alpha_2 \Leftrightarrow A_{\alpha_1} \subseteq A_{\alpha_2}$, $\alpha_1, \alpha_2 \in [0,1]$. Es inmediato que $A_0 = U$.

Teorema A.1. *(Teorema de Representación) Si $\widetilde{A}$ es un conjunto fuzzy y A_α sus α-cortes, $\alpha \in [0,1]$, se verifica que:*

$$\widetilde{A} = \bigcup_{\alpha \in [0,1]} \alpha A_\alpha$$

entendiendo esta notación formal como la igualdad entre las funciones de pertenencia de ambos conjuntos. Si $\mu_{A_\alpha}(x)$ nota la función característica de A_α, caso particular de la función de pertenencia,

$$\mu_{A_\alpha}(x) = \begin{cases} 1 & \textit{si y sólo si } x \in A_\alpha \\ 0 & \textit{en otro caso} \end{cases}$$

la función de pertenencia del conjunto fuzzy $\widetilde{A}$ puede expresarse en términos de las funciones características de sus α-cortes según la fórmula

$$\mu_A(x) = \sup_{\alpha \in [0,1]} min(\alpha, \mu_{A_\alpha}(x))$$

□

Definición A.6. Se denomina *conjunto de niveles* de un conjunto fuzzy $\widetilde{A}$, al conjunto de grados de pertenencia de sus elementos:

$$L(\widetilde{A}) = \{a \ / \ \mu_{\widetilde{A}}(x) = a, \quad x \in U\}$$

□

Definición A.7. Un conjunto fuzzy es convexo si y sólo si sus α-cortes son convexos.

□

Una definición equivalente a la convexidad es la siguiente:

Definición A.8. Un conjunto fuzzy $\widetilde{A}$ es convexo si y sólo si

$$\forall x_1, x_2 \in X, \ \forall \lambda \in [0,1], \quad \mu_{\widetilde{A}}(\lambda x_1 + (1 \quad \lambda) x_2) \geq min(\mu_{\widetilde{A}}(x_1), \mu_{\widetilde{A}}(x_2))$$

□

Definición A.9. Un conjunto fuzzy se dice normalizado si y sólo si $\exists x \in U$ en el que $\mu_{\widetilde{A}}(x) = 1$.

□

Esta definición implica que en los conjuntos fuzzy normalizados la $h(\widetilde{A}) = 1$.

Definición A.10. Se denomina *cardinalidad* de un conjunto fuzzy $\widetilde{A}$, al área que define (en caso discreto, la suma de sus grados de pertenencia):

$$|\widetilde{A}| = \int_U \mu_A(x) dx$$

□

Definición A.11. Se denomina *cardinalidad relativa* (respecto a un conjunto de referencia dado) de un conjunto fuzzy $\widetilde{A}$ respecto a un conjunto de referencia dado a:

$$||\widetilde{A}|| = \frac{|\widetilde{A}|}{U}$$

□

A.2.1.3. Operaciones con Conjuntos Fuzzy

Al igual que en la lógica tradicional, las operaciones lógicas que se pueden establecer entre conjuntos fuzzy son la intersección, la unión y el complemento (negación). Mientras que el resultado de operar dos conjuntos clásicos es un nuevo conjunto clásico, las mismas operaciones con conjuntos fuzzy nos darán como resultado otros conjuntos también fuzzy.

Hay muchas formas de definir estas operaciones. Cualquier operación que cumpla las propiedades de una t-norma puede ser usada para hacer la intersección, de igual manera que cualquier operación que cumpla las propiedades de una t-conorma puede ser empleada para la unión.

Las operaciones se definen de la siguiente manera:

- Intersección: $\widetilde{A} \cap \widetilde{B} = \{(x, \mu_{\widetilde{A} \cap \widetilde{B}}) \ / \ \mu_{\widetilde{A} \cap \widetilde{B}}(x) = T[\mu_{\widetilde{A}}(x), \mu_{\widetilde{B}}(x)]\}$
- Unión: $\widetilde{A} \cup \widetilde{B} = \{(x, \mu_{\widetilde{A} \cup \widetilde{B}}) \ / \ \mu_{\widetilde{A} \cup \widetilde{B}}(x) = S[\mu_{\widetilde{A}}(x), \mu_{\widetilde{B}}(x)]\}$
- Complemento: $\mu_{\neg \widetilde{A}}(x) = 1 \quad \mu_{\widetilde{A}}(x)$

En la Tabla A.1 se muestran las propiedades que deben cumplir estas familias de funciones y algunos ejemplos.

	Propiedades	Ejemplos
T-Normas $T : \|0,1\| \times \|0,1\| \rightarrow \|0,1\|$ $\mu_{A \cap B}(x) = T\|\mu_A(x), \mu_B(x)\|$	**Conmutativa:** $T(a,b) = T(b,a)$ **Asociativa:** $T(a, T(b,c)) = T(T(a,b), c)$ **Monotonía:** $T(a,b) \geq T(c,d)$ si $a \geq c$ y $b \geq d$ **Condiciones frontera:** $T(a,1) = a$ y $T(a,0) = 0$	**Intersección estandar** $T(a,b) = min(a,b)$ **Producto algebraico** $T(a,b) = a \cdot b$ **Intersección drástica** $T(a,b) = \begin{cases} a & \text{si } b = 1 \\ b & \text{si } a = 1 \\ 0 & \text{en otro caso} \end{cases}$ **Familia de Yager:** con $p > 0$ $T(a,b) = 1 - min(1, \sqrt[p]{(1-a)^p + (1-b)^p})$
T-Conormas (S-normas) $S : \|0,1\| \times \|0,1\| \rightarrow \|0,1\|$ $\mu_{A \cup B}(x) = S\|\mu_A(x), \mu_B(x)\|$	**Conmutativa:** $S(a,b) = S(b,a)$ **Asociativa:** $S(a, S(b,c)) = S(S(a,b), c)$ **Monotonía:** $S(a,b) \geq S(c,d)$ si $a \geq c$ y $b \geq d$ **Condiciones frontera:** $S(a,0) = a$ y $S(a,1) = 1$	**Unión estandar** $S(a,b) = max(a,b)$ **Suma algebraica** $S(a,b) = a + b - a \cdot b$ **Unión drástica** $S(a,b) = \begin{cases} a & \text{si } b = 0 \\ b & \text{si } a = 0 \\ 1 & \text{en otro caso} \end{cases}$ **Familia de Yager:** $T(a,b) = min(1, \sqrt[p]{(a^p + b^p)})$ con $p > 0$

Negación (Complemento) $N : \|0,1\| \rightarrow \|0,1\|$ $\mu_{\neg \widetilde{A}}(x) = N\|\mu_{\widetilde{A}}(x)\|$	**Monotonía:** N es no creciente **Condiciones frontera:** $N(0) = 1$ y $N(1) = 0$	**Negación original** $N(a) = 1 - a$ **Involutivas** $N(a) = \sqrt[w]{1 - a^w}$, $w \in (0, \infty\|$ **No Involutivas** $N(a) = \begin{cases} 1 & \text{si } x < a \\ 0 & \text{si } x \geq a \end{cases}$ con $a \in \|0,1\|$

Tabla A.1: t-normas, t-conormas y negación.

En la figura A.2 podemos ver una representación gráfica de dichas operaciones.

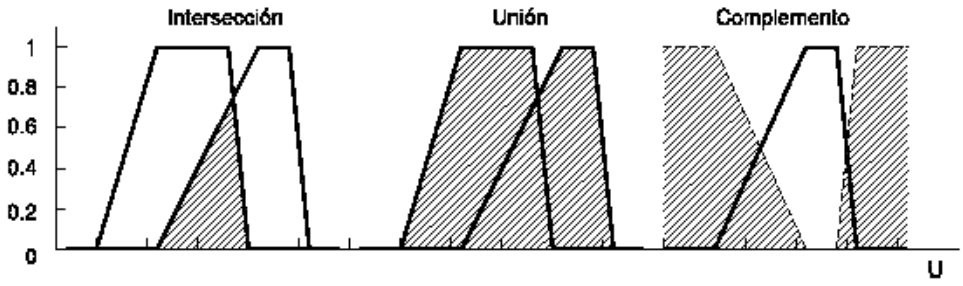


Figura A.2: Intersección, Unión y Complemento en conjuntos fuzzy.

A.2.1.4. Medidas de Similaridad

Las medidas de similaridad entre conjuntos fuzzy ha ganado atención de los investigadores en un amplio campo de aplicaciones del mundo real. Basadas en medidas de similaridad que son muy útiles en campos como el aprendizaje computacional, sistemas de decisión, predicción, etc, se han propuesto muchas medidas de similaridad entre conjuntos fuzzy.

Una medida de similaridad es una función que asocia un valor numérico a una sucesión de conjuntos, con la idea de que un valor más alto indica similitud más grande entre ellos. Una medida de similaridad es un tipo de función *"scoring"*.

Algunas de las medidas de similaridad utilizadas en la memoria son las siguientes, [170]:

Definición A.12. Sean $\widetilde{A}$ y $\widetilde{B}$ dos conjuntos fuzzy sobre el dominio U. La medida de similaridad S_1 se define como:

$$S_1(\widetilde{A}, \widetilde{B}) = \frac{||\widetilde{A} \cap \widetilde{B}||}{||\widetilde{A} \cup \widetilde{B}||}$$

donde $||\zeta||$ es la cardinalidad relativa del conjunto fuzzy ζ sobre U, $\cap$ denota el mínimo y $\cup$ el máximo. □

Definición A.13. Sean $\widetilde{A}$ y $\widetilde{B}$ dos conjuntos fuzzy sobre el dominio U. La medida de similaridad S_2 se define como:

$$S_2(\widetilde{A}, \widetilde{B}) = 1 \quad ||\widetilde{A} \nabla \widetilde{B}||$$

donde $||\nabla||$ es la diferencia simétrica definida por la distancia de Hamming $\mu_{A \nabla B}(u) = |\mu_A(u) \quad \mu_B(u)|$. □

Definición A.14. Sean $\widetilde{A}$ y $\widetilde{B}$ dos conjuntos fuzzy sobre el dominio U. La medida de similaridad S_3 se define como:

$$S_3(\widetilde{A}, \widetilde{B}) = 1 \quad ||\widetilde{A} \triangle \widetilde{B}||$$

donde $\mu_{A \triangle B}(u) = max\{\mu_{A \cap \neg B}(u), \mu_{\neg A \cap B}(u)\}$. □

A.2.1.5. Operadores de agregación

Los operadores de agregación o combinación fuzzy han sido ampliamente usados y estudiados desde el mismo origen de la lógica fuzzy. Dubois y Prade presentan en [103] una clasificación de los operadores fuzzy.

- Operadores menores o iguales que el mínimo. Exigen que todos y cada uno de los criterios que se desean combinar se cumplan de forma simultánea. La combinación resultante estará acotada por el grado de pertenencia menor. A estos operadores, entre los que se incluyen las T-normas, se les conoce como operadores de intersección.
- Operadores acotados por el máximo. El resultado está acotado inferiormente por el valor del mayor grado de pertenencia. A estos operadores, entre los que se incluyen las T-conormas, se les conoce como operadores de unión.
- Operadores promedio. Describen un comportamiento de compensación, o de promedio, entregando un resultado que está entre los valores máximo y mínimo.

Entre todos los posibles operadores, describimos la integral fuzzy y los operadores OWA.

▶ Integrales Fuzzy

Las integrales fuzzy constituyen una amplia familia de operadores de agregación, en la que se modela la interacción entre los valores a agregar. En [133] la integral fuzzy ha sido aplicada a la combinación de clasificadores.

Sea $X = \{x_1, ..., x_n\}$ con $x_i \in [0, 1]$ el conjunto de valoraciones que se quieren agregar.

Definición A.15. Una medida fuzzy $\mu(\cdot)$ definida sobre X es una función $\mu(\cdot) : P(X) \to [0,1]$ cumpliendo los siguientes propiedades:

i) $\mu(\emptyset) = 0$, $\mu(X) = 1$.

ii) $\widetilde{A} \subseteq \widetilde{B} \Rightarrow \mu(A) \leq \mu(B)$.

$P(X)$ indica el conjunto potencia de X (el conjunto de todos los subconjuntos de X). □

Una medida sobre X necesita 2^n coeficientes para que se pueda definir, los cuales son los valores de μ para todos los diferentes conjuntos de X.

Las integrales fuzzy son integrales de una función real con respecto a una medida fuzzy, por analogía con la integral de Lebesgue la cual está definida con respecto a una medida aditiva arbitraria. Hay distintas definiciones de integrales fuzzy, entre las cuales las más representativas son la de Sugeno y la de Choquet.

Definición A.16. Sea μ una medida fuzzy sobre X. La integral de Choquet discreta de una función $f : X \to R^+$ con respecto a μ se define como

$$C_\mu(f(x_1), \ldots, f(x_n)) = \sum_{i=1}^{n} (f(x_{(i)}) \quad f(x_{(i\ \ 1)}))\mu(A_{(i)})$$

donde $\cdot_{(i)}$ indica que los índices se han permutado tal que $0 \leq f(x_{(1)}) \leq \ldots f(x_{(n)}) \leq 1$; $A_{(i)} = \{x_{(i)}, \ldots, x_{(n)}\}$, y $f(x_{(0)}) = 0$. □

Definición A.17. Sea μ una medida fuzzy sobre X. La integral de Sugeno discreta de una función $f : X \to [0,1]$ con respecto a μ se define como

$$S_\mu(f(x_1), \ldots, f(x_n)) = \bigvee_{i=1}^{n} (f(x_{(i)})) \wedge \mu(A_{(i)})$$

donde $\cdot_{(i)}$ indica que los índices se han permutado tal que $0 \leq f(x_{(1)}) \leq \ldots f(x_{(n)}) \leq 1$; $A_{(i)} = \{x_{(i)}, \ldots, x_{(n)}\}$, y $f(x_{(0)}) = 0$. □

▶ Operadores OWA

La familia de operadores OWA incluyen el máximo, el mínimo y la media. Han sido usados en variadas áreas, tales como la toma de decisiones, el reconocimiento de patrones y la recuperación de información.

Los operadores OWA (*Ordered Weighted Averaging*) fueron introducidos por Yager en 1988, [262]. Los operadores OWA son conocidos como operadores de compensación y son

operadores de agregación de información numérica que tiene en cuenta el orden de las valoraciones que van a ser agregadas. Los operadores OWA son similares a los operadores de medias ponderadas con la diferencia que el peso asociado a un criterio específico no afecta su valor sino la posición que ocupa, de allí el nombre "ordenados". Cada peso w_i se asocia con el i-ésimo elemento más grande independientemente de su valor.

Definición A.18. *Operador OWA.* Sea $A = \{a_1, ..., a_n\}$ con $a_i \in [0,1]$ el conjunto de valoraciones que se quieren agregar y $W = (w_1, \ldots, w_n)$ su vector de pesos asociado, tal que (i) $w_i \in [0,1]$, con $1 \leq i \leq n$ y (ii) $\sum_{i=1}^{n} w_i = 1$; el operador OWA, F, se define como:

$$F(a_1, \ldots, a_n) = \sum_{j=1}^{n} w_j \cdot b_j$$

donde b_j es el j-ésimo mayor valor del conjunto A. Por tanto, a partir de los elementos de A podemos obtener un conjunto B ordenando dichos elementos en orden decreciente, es decir,

$$B = \{b_1, \ldots, b_n\} \mid b_i \geq b_j \; si \; i < j$$

y definir el operador OWA de la siguiente forma:

$$F(a_1, ..., a_n) = W \cdot B$$ □

En los operadores OWA es fundamental el paso de la reordenación. Un agregado x_i no se asocia a un peso particular w_j. Los pesos están asociados a una *posición ordenada* j particular de los argumentos. Se pueden obtener un gran número de operadores según la elección de los pesos. Por esto, los operadores OWA son conocidos como familia de operadores OWA. Por ejemplo:

1. Si $W = [1, 0, ..., 0]^t$ entonces $F(x_1, ...x_n) = max(x_1, ...x_n)$
2. Si $W = [0, 0, ..., 1]^t$ entonces $F(x_1, ...x_n) = min(x_1, ...x_n)$
3. Si $W = [1/n, 1/n, ..., 1/n]^t$ entonces $F(x_1, ...x_n) = 1/n \sum_{i=1}^{n} x_i)$

Los operadores OWA permiten definir una medida de optimismo $orness(W) = \frac{1}{n-1} \sum_{i=1}^{n}((n - i)w_i)$. Esta medida determina el grado en que la agregación es parecida a una operación OR (max):

1. orness $([1, 0, ..., 0]^t) = 1$. El operador *máximo* presenta el mayor grado de optimismo.
2. orness $([0, 0, ..., 1]^t) = 0$. El operador *mínimo* presenta el menor grado de optimismo.
3. orness $([1/n, 1/n, ..., 1/n]^t) = 1/2$. El operador *promedio* presenta un grado de optimismo neutral.

Por último diremos que en muchos casos es más adecuado utilizar una agregación con pesos paramétricos basados en la información que el decisor posee de los pesos cuando el conocimiento de la situación real y el sentido común así lo indican, [67].

A.3. Modelado Lingüístico Fuzzy

La información que manejamos en el mundo real puede tener diferentes rangos de valoración y los valores pueden tener distinta naturaleza. En ocasiones, puede que no sea fácil valorarla de forma precisa mediante un valor cuantitativo, sin embargo puede que sí sea factible hacerlo de forma cualitativa. En este caso, adoptar un enfoque lingüístico suele ofrecer mejores resultados que si aplicamos uno numérico. Este hecho se puede deber a:

- Situaciones en las que la información, por su propia naturaleza, no puede ser cuantificada y por tanto únicamente puede ser valorada mediante el uso de términos lingüísticos, como sucede cuando realizamos una valoración sobre un libro que hayamos leído, que solemos usar términos como *bueno*, *regular* o *malo*.
- En otros casos, trabajar con información precisa de forma cuantitativa no es posible, o bien porque no están disponibles los elementos necesarios para llevar a cabo una medición exacta de esa información, o bien porque el coste computacional es demasiado alto y nos basta con la aplicación de un valor aproximado. Por ejemplo, cuando evaluamos la velocidad de una motocicleta, en lugar de usar valores numéricos, solemos usar términos tales como *rápida*, *muy rápida* o *lenta*.

A.3.1. Variables lingüísticas

El modelado lingüístico fuzzy es, pues, un enfoque aproximado basado en la Teoría de Conjuntos Fuzzy. Este modelo representa los aspectos cualitativos como valores lingüísticos mediante lo que se conoce como *variables lingüísticas*, [267–269]. Una variable lingüística se caracteriza por un *valor sintáctico* o *etiqueta* que es una palabra o frase perteneciente a un conjunto de términos lingüísticos, y por un *valor semántico* o *significado* de dicha etiqueta que viene dado por un subconjunto fuzzy en un universo de discurso. Formalmente se define de la siguiente manera.

Definición A.19. [267–269] Una *variable lingüística* está caracterizada por una 5-tupla $(H, TL(H), U, G, SM)$, donde:

- H es el nombre de la variable;
- $TL(H)$ (o sólo TL) simboliza el conjunto de términos lingüísticos de H, es decir, el conjunto de nombres de valores lingüísticos de H, donde cada valor es una variable fuzzy denotada genéricamente como X que toma valores en el universo de discurso;
- U el universo de discurso que está asociado con una variable base denominada u;
- G es una regla sintáctica (que normalmente toma forma de gramática) para generar los nombres de los valores de H;

- SM es una regla semántica para asociar significado a cada elemento de $TL(H)$, que será un subconjunto fuzzy de U. □

Por ejemplo, consideremos la variable lingüística $H = velocidad$ y la variable base $u \in U$ con $U = [0, 125]$. El conjunto de términos asociados con la velocidad podría ser $Ter(H) = \{baja, media, alta\}$ donde cada término en $Ter(H)$ es el nombre de un valor lingüístico de *velocidad* (fig. A.3). El significado $SM(L)$ de una etiqueta $L \in Ter(H)$ se define como la restricción $L(u)$ sobre la variable base u impuesta según el nombre de L. Por lo tanto $SM(L)$ es un conjunto fuzzy de U cuya función de pertenencia $L(u)$ representa la semántica del nombre L.

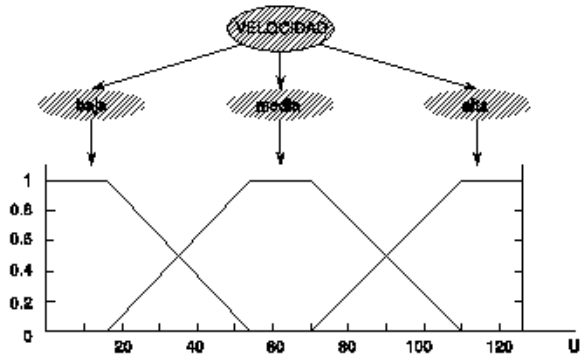


Figura A.3: Ejemplo de variable lingüística.

Un aspecto importante que es necesario analizar con el fin de establecer la descripción de una variable lingüística es la *granularidad de la incertidumbre*, [34], es decir, la cardinalidad del conjunto de términos lingüísticos usado para expresar y representar la información. La cardinalidad debe ser suficientemente baja como para no imponer una precisión excesiva en la información que se quiera expresar y suficientemente alta como para conseguir una discriminación de las valoraciones en un número limitado de grados. Habitualmente la cardinalidad usada en los modelos lingüísticos suele ser un valor impar, como 7 ó 9, no superando las 11 ó 13 etiquetas. Una vez establecida la cardinalidad del conjunto de términos lingüísticos, hay que definir dicho conjunto, es decir, cuáles van a ser las etiquetas lingüísticas y su semántica asociada.

Por tanto, ante cualquier problema que hace uso de información lingüística, el primer objetivo que hay que satisfacer es la elección de los términos lingüísticos con sus correspondientes semánticas, para así establecer el conjunto de etiquetas que se va a usar. En la literatura podemos encontrar dos posibilidades distintas:

- Podemos asumir que todos los términos del conjunto de etiquetas son igualmente

informativos, es decir, que estén distribuidos simétricamente tal y como sucede en los modelados lingüísticos difusos ordinales, 2-tupla y multigranulares, [147, 149, 150].

- Podemos asumir que no todos los términos del conjunto de etiquetas son igualmente informativos, es decir, las etiquetas no están distribuidas simétricamente. En este caso, necesitamos un enfoque lingüístico difuso no balanceado, [148], para gestionar los conjuntos de términos lingüísticos con distintos niveles de discriminación a ambos lados del término medio.

  En la figura A.4 podemos ver una representación gráfica de una partición no balanceada de un dominio.

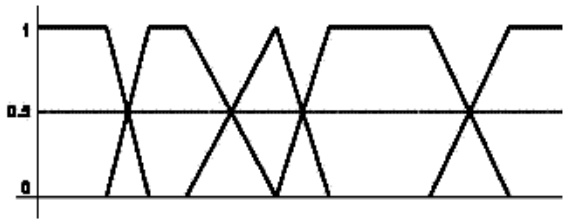


Figura A.4: Partición de un dominio en etiquetas lingüísticas.

A.4. Sistemas de control fuzzy

A.4.1. Reglas de control fuzzy

Los sistemas de control fuzzy permiten describir el conjunto de reglas que utilizaría un ser humano que controlase el proceso, con toda la imprecisión que poseen los lenguajes naturales y, sólo a partir de estas reglas, generan las acciones que realizan el control. Por esta razón, también se les denominan controladores lingüísticos.

El sistema de control esta dado como un conjunto de expresiones de la forma "SI condiciones ENTONCES acciones" en las apareceran variables que toman valores lingüísticos. Por ejemplo IF x_1 es pequeño AND x_2 es grande THEN y es medio.

Estas expresiones son las reglas de control fuzzy difuso. La parte izquierda de la regla es el antecedente y la parte derecha es el consecuente. En general las variables de condición x_1, x_2, ..., son las entradas del sistema de control y la variable y es la salida del mismo. Las palabras pequeño, grande y medio son valores fuzzy que toman las variables fuzzy x_1, x_2 e y que se definen mediante conjuntos fuzzy. Los controladores fuzzy están formados por grupos de reglas y actúan de la forma siguiente: Cuando se les proporciona el valor actual de las variables de entrada se obtiene el valor de las variables de salida, calculado mediante un método de inferencia fuzzy. Teniendo en cuenta que los sistemas de control deben actuar en tiempo real, los métodos de inferencia que se usan tienen que ser sencillos y rápidos.

A.4.2. Métodos de inferencia

Sea cual sea la forma de las variables y reglas, el proceso de inferencia se realiza como:

1. Calcular la compatibilidad entre los valores actuales de las entradas y los antecedentes de las reglas.
2. Encontrar los resultados de las inferencias de cada regla (en qué grado se verifica).
3. Encontrar el resultado de la inferencia completa como un promedio de los resultados de las inferencias de cada regla.

A.4.3. Inferencia fuzzy

La Inferencia lógica consiste en la combinación de proposiciones y reglas para producir nuevas proposiciones. Así, al combinar la proposición "X es A" con la regla "IF X es A THEN Y es B" se puede inferir la proposición "Y es B". Como se ve, las reglas expresan un tipo de relación entre dos o más variables fuzzy y representa una implicación lógica, donde el conocimiento se expresa por reglas. En general el mecanismo de inferencia tiene la forma señalada, entonces la inferencia para la lógica fuzzy tendrá la misma forma, teniendo su principal diferencia en que dos proposiciones no necesariamente deben ser idénticas, debido a que las fronteras de los conjuntos no son precisas. Así, al combinar la proposición "X es A*" con la regla "IF X es A THEN Y es B", puede obtenerse la proposición "Y es B*".

A.4.4. Fuzzificación

En esta etapa se transforma las variables de entrada del modelo (u) en variables difusas, donde a cada variable de entrada se le asigna un grado de pertenencia a cada uno de los conjuntos difusos que se han considerado, mediante las funciones de membresía asociadas a estos conjuntos difusos. Para esta interfaz se deben tener definidos los rangos de variación de las variables de entrada y los conjuntos difusos asociados con sus respectivas funciones de pertenencia. Las variables de entrada son valores concretos de las variables de entrada y las salidas son grados de pertenencia a los conjuntos difusos considerados.

A.4.5. Base de Conocimiento

Contiene las reglas lingüísticas del control y la información referente a las funciones de pertenencia de los conjuntos difusos. Como ya se ha visto en la parte de inferencia difusa, las reglas tienen la forma "Si u es A y v es B entonces y es C" donde A, B y C son los conjuntos difusos de las variables de entrada u y v, y de la variable de salida y respectivamente. Existen varias formas de derivar las reglas, entre las que destacan las basadas en:

- La experiencia de expertos y el conocimiento de ingeniería de control. La base de reglas se determina a partir de entrevistas con el operador o a través del conocimiento de la dinámica del proceso.
- La modelación del proceso. Los parámetros de la base de conocimiento se obtienen a partir de datos de entrada y salida del proceso.

A.4.6. Motor de Inferencia

El motor de inferencia usan los principios de la lógica difusa acerca de la inferencia difusa, para realizar un mapeo de los conjuntos difusos de entrada a los conjuntos difusos de salida. Cada regla es interpretada como una implicación difusa. Es decir el bloque de inferencia es aquel en el cual se realiza la "traducción matemática" de las reglas difusas: estas reglas modelan el sistema, pero para poder trabajar con ellas y extraer un resultado, se debe evaluar matemáticamente la información que reflejan. Esta etapa realiza la tarea de calcular las variables de salida a partir de las variables de entrada, mediante las reglas del controlador y la inferencia difusa, entregando conjuntos difusos de salida. Este bloque mediante mecanismos de inferencia relaciona conjuntos difusos de entrada y de salida. La secuencia de cálculos que realiza el motor de inferencia incluye:

- Determinar el grado de cumplimiento de cada regla a partir de los grados de pertenencia de las variables de entrada obtenidos en la etapa de fusificación. Debido a que las premisas de la reglas están unidas por operadores AND, definidas como la intersección de conjuntos difusos.
- Como se sabe, para cada regla se tiene una consecuencia, que tiene asociado una función de pertenencia. Por lo tanto a la salida se tendrá un conjunto difuso de salida representado por su respectiva función de pertenencia.
- Para evaluar el conjunto total de reglas, se unen los conjuntos difusos resultantes de cada regla, generándose un conjunto de salida que estará representada por una norma. De esta forma, se obtiene una salida difusa del controlador, con una función de pertenencia.

A.4.7. Defuzzificación

En este bloque a partir del conjunto difuso obtenido en el bloque de inferencia y mediante métodos matemáticos de defusificación, se obtiene el valor concreto de la variable de salida. Este elemento provee salidas discretas y determinísticas a partir de los conjuntos difusos obtenidos como resultado de la inferencia.

A.4.7.1. Métodos de Defuzzificación

El bloque defuzzificador tiene como entrada el conjunto difuso de salida, resultado del bloque de inferencia, y_{out}, y la salida es un valor concreto de la variable de salida, y^*. Para obtener a partir del conjunto difuso de salida que resulta la agregación de todas las reglas, un resultado escalar, se aplican métodos matemáticos. Entre los más utilizados, [162], se encuentran los métodos: Centroide, Centro de Sumas, Media de Máximos y Promedio de los Centros.

Método del Centroide o Centro de Areas (COA): este método calcula y^* como el centro geométrico del valor difuso de la salida y_{out}. En un dominio discreto se calcula como:

$$y^* = \frac{\sum_{i=1}^{N} y_i \mu_{out}(y_i)}{\sum_{i=1}^{N} \mu_{out}(y_i)}$$

donde la suma se extiende a los N valores y_i del dominio discreto. En el caso de que tratarse de un dominio continuo, el sumatorio se convierte en una integral.

Este método tiene a favorecer los valores centrales y además no tiene en cuenta los solapamientos, puesto que opera sobre y_{out} después de la agregación.

Método del Centro de Sumas (COS): este método soluciona el problema de ignorar las áreas solapadas, puesto que opera separadamente sobre cada contribución. Para un dominio discreto se define como:

$$y^* = \frac{\sum_{i=1}^{N} y_i \sum_{k=1}^{M} \mu_{C'_k}(y_i)}{\sum_{i=1}^{N} \sum_{i=1}^{M} \mu_{C'_k}(y_i)}$$

donde $\mu G'_k(y)$ es la función de pertenencia resultante de aplicar la regla k.

Método de la Media de los Máximos (MOM): este método calcula y^* como el promedio de los puntos de máximo grado de pertenencia en la salida global $\mu_{out}(y)$. Esta técnica es muy rápida y no muestra preferencia por los valores centrales, como si ocurre con COA y COS. Sin embargo, no tiene en cuenta la forma de $\mu_{out}(y)$. Se calcula como:

$$y^* = \sum_{j=1}^{P} \frac{u_j}{P}$$

donde u_j es el j-ésimo máximo de y_{out} y P el número total de máximos.

Promedio de centros (CA): este método opera separadamente sobre cada contribución y calcula un promedio ponderado de los centros de las contribuciones. Se calcula como:

$$y^* = \frac{\sum_{k=1}^{M} \bar{yk} \mu_{C'_k}}{\sum_{k=1}^{M} \mu_{C'_k}}$$

donde $\mu G'_k(y)$ es la función de pertenencia resultante de aplicar la regla k y $\bar{yk}$ es el centro del conjunto difuso G'_k.

Apéndice B

Agentes Software y Sistemas Multiagente

B.1. Introducción

Los agentes y sistemas multiagente, son una herramienta muy útil a la hora de modelar un sistema en el que distintas entidades interaccionen. Por tanto, los consideramos un instrumento indispensable a la hora de plantear el marco general de diseño y construcción de metaheurísticas cooperativas que se desarrolla en esta tesis. En particular en este apéndice nos centraremos en aquellos aspectos más relevantes para nuestros objetivos, dando una visión global de los agentes y sistemas multiagente, y prestando especial atención a los esquemas de comunicación y en concreto al esquema de pizarra que es el que consideramos más adecuado para nuestros propósitos.

B.2. El paradigma de Agente Software

B.2.1. Concepto general de Agente

Los primeros artículos que contienen la idea de agente software aparecen alrededor de 1980, en los cuales se plantea la cooperación entre sistemas expertos para la resolución de problemas complejos. Estos sistemas expertos se diseñan como aplicaciones basadas en conocimiento que permiten resolver problemas en dominios específicos a través de procesos de razonamiento [144]. Sin embargo, pronto se detecta que su capacidad de resolución es muy limitada y que se necesita intercambiar conocimiento entre ellos para paliar estas limitaciones. Surge entonces la idea de tomar cada sistema experto como un *agente de colaboración* con otros sistemas con el fin de aumentar su capacidad individual de resolución de problemas. A partir de ese momento, la tecnología de agente ha ido avanzando progresivamente mediante la adopción de técnicas de diversos campos tales como Inteligencia Artificial, Sistemas Distribuidos e

Ingeniería del Software.

Junto a esta continua evolución han ido apareciendo múltiples definiciones de agente software. Las definiciones más utilizadas para describir el estado actual de esta tecnología pueden encontrarse en los artículos de Wooldridge [261] y Luck [182], donde se recogen los siguientes aspectos:

> Un agente software es un programa computacional que actúa en representación de una *entidad* y en un determinado *entorno* para conseguir ciertos *objetivos* de una forma *autónoma* e *inteligente*, y en la que posiblemente necesite *interaccionar* con otros agentes.

Comentemos de forma breve los conceptos que aparecen en esta definición. En primer lugar, la *entidad* representada por el agente puede ser una persona, sistema o aplicación. El *entorno* donde actúa el agente se corresponden con los elementos que lo rodean, tales como otros agentes, recursos, entidades que manejan información, etc. Este entorno suele ser bastante dinámico debido a los frecuentes cambios de estado de los elementos que lo componen. En cuanto a los *objetivos* del agente, éstos coinciden en un primer nivel con los objetivos de la entidad a la que representan.

El funcionamiento del agente de forma *autónoma* se refiere a que la persecución de un objetivo se hace de forma continua, robusta y adaptable. Esto significa que tras la invocación inicial en la que se especifica el objetivo a conseguir, el agente es capaz de adaptarse a los cambios del entorno y de buscar diferentes soluciones para alcanzar dicho objetivo sin necesidad de nuevas invocaciones. Este funcionamiento autónomo va ligado a la *inteligencia* del agente, que consiste en su capacidad de razonamiento y aprendizaje. Finalmente, el concepto de *interactuación* se refiere a la habilidad del agente para comunicarse y coordinarse con otros agentes a través de ciertos lenguajes y protocolos.

Nótese que la definición de agente tiene un carácter general para dar cabida a todos los tipos de agentes que se han ido desarrollando. Aplicando ciertos criterios sobre algunos de los aspectos que hemos resaltado en esta definición general podemos obtener una clasificación más específica de agentes. El siguiente apartado muestra una revisión de tipos de agentes.

B.2.2. Tipos de Agentes

Los criterios para clasificar agentes vienen dados por los diferentes ángulos desde los que pueden ser observados. Algunas propuestas para la definición de criterios pueden consultarse en [35, 204, 261]. En este documento, los criterios considerados más importantes para un agente son tres: su comportamiento individual, el modo de interacción con otros agentes y su utilidad.

a) Tipos de agentes según su comportamiento individual

Este criterio clasifica a los agentes según la manera de actuar a partir de la información recibida de su entorno. Antes de explicar estos tipos de agentes, podemos concebir una arquitectura abstracta común para todos ellos con el fin de mostrar qué se entiende por el comportamiento de un agente. La arquitectura está formada por tres módulos: Percepción, Control y Acción. En primer lugar, el agente obtiene información del entorno a través del módulo de percepción. Este módulo procesa la información de forma que pueda ser interpretada por el módulo de control, el cual se encarga de dirigir el comportamiento del agente ante la información recibida. En este segundo módulo se establecen las distinciones entre agentes: según las características del procesamiento de la información recibida tendremos diferentes tipos de agentes. Finalmente, la actuación generada por el módulo de control es transferida al módulo de acción encargado de llevarla a cabo. Estas acciones pueden ser tanto internas como externas, donde las primeras afectan al propio módulo de control del agente mientras que las segundas inciden sobre el entorno.

Teniendo en cuenta la implementación del módulo de control, podemos distinguir dos tipos de agentes: *reactivos* y *racionales*. A continuación los veremos con más detalle.

Agentes reactivos [33, 94]. El modelo de computación de estos agentes está basado en el ciclo *recepción de eventos/reacción*. La recepción de eventos está gestionada por el módulo de percepción, mientras que la reacción se divide en dos fases. En la primera, el módulo de control selecciona un procedimiento sencillo para reaccionar ante el evento recibido teniendo en cuenta el estado interno del agente. En la segunda, este procedimiento es ejecutado por el módulo de acción, dando lugar a cambios en dicho estado interno o a acciones sobre el entorno.

La arquitectura típica del módulo de control de los agentes reactivos puede observarse en la Figura B.1. Esta arquitectura está compuesta por varios módulos de comportamiento donde cada uno contiene un procedimiento básico para resolver una tarea específica. Estos comportamientos son activados según el evento recibido y el estado interno del agente. En caso de activarse más de un módulo de comportamiento es necesario decidir cuál es el más adecuado según la situación, inhibiendo el resto de comportamientos. Otra posibilidad es fusionar comportamientos mediante una ponderación de éstos. En cualquier caso, observar que un agente reactivo no tiene mecanismos explícitos para representar conocimiento ni utiliza procesos de razonamiento sobre la información recibida. La inteligencia en estos agentes no es un propiedad individual de cada uno, sino que emerge de la interacción entre ellos cuando se comportan como una comunidad.

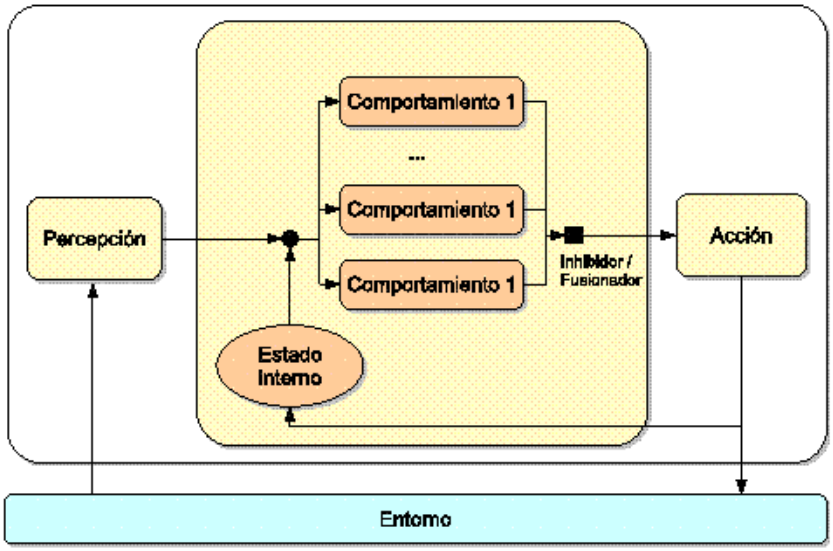


Figura B.1: Arquitectura del módulo de control en agente reactivos.

Las ventajas de los agentes reactivos radican en la sencillez para definir comportamientos y en sus respuestas prácticamente instantáneas ante la llegada de eventos. Por tanto, su aplicación está orientada a entornos impredecibles y altamente cambiantes donde el agente tiene que dar una respuesta rápida y eficaz. En cambio, tanto la falta de un modelo para representar conocimiento como la ausencia de procesos de razonamiento desaconsejan su uso en problemas donde el agente necesita procesar la información recibida y planificar las acciones que se tienen que llevar a cabo.

Agentes racionales [190,228]. También llamados *cognitivos*, su modelo de computación se basa en percibir información, integrarla en un modelo de conocimiento, razonar sobre dicho modelo y actuar según este razonamiento. Para alcanzar sus objetivos, los agentes racionales emplean procesos de razonamiento sobre el modelo de conocimiento, y en algunos casos utilizan procesos de planificación y/o aprendizaje. Como resultado de estos procesos se obtienen las acciones que debe realizar el agente acompañadas siempre de una justificación. Estas acciones pueden modificar los objetivos del agente e incidir sobre el entorno.

El módulo de control que gestiona los procesos de integración de conocimiento y razonamiento está diseñado mediante una arquitectura deliberativa, la cual puede observarse en la Figura B.2. Una de las piezas fundamentales de esta arquitectura es el conocimiento del agente. Dicho modelo se obtiene mediante formalismos computacionales tales como esquemas conceptuales combinados con programas de reglas. Otro componente básico de la arquitectura es el módulo de razonamiento lógico. Este modulo toma el conocimiento y los objetivos del agente como entrada para obtener las acciones que hay que llevar a cabo. El resultado

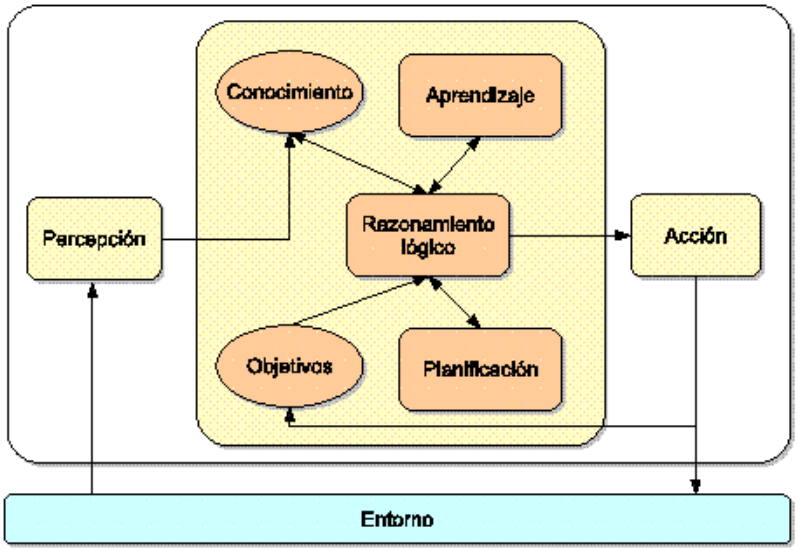


Figura B.2: Arquitectura del módulo de control en agente racionales.

obtenido suele contener un plan con las acciones a ejecutar.

Debido a sus características, los agentes racionales están especialmente indicados para resolver problemas complejos basados en conocimiento (e.g., ayuda a la toma de decisiones). Sin embargo, su implementación no es una tarea sencilla y necesita de ciertas limitaciones para poder llevarla a la práctica, tales como el uso de procesos de razonamiento que ofrezcan respuestas en un tiempo adecuado.

Finalmente, las arquitecturas de los agentes reactivos y racionales pueden combinarse dando lugar a arquitecturas *híbridas*. De esta forma es posible tener un agente que implemente ambos enfoques y los utilice según las características del problema y del entorno. La forma más común de combinar ambas arquitecturas es mediante una estructura de capas, donde las inferiores contienen la arquitectura reactiva y las superiores contienen la arquitectura racional. Además, el módulo de control decide en todo momento qué enfoque es más adecuado para la tarea que se está resolviendo.

b) Tipos de agentes según su modo de interacción

El criterio de interacción entre agentes permite clasificar dichos agentes como entidades individuales que tienen un rol dentro de una organización. Así, podemos hacer una primera distinción entre agentes **individualistas** y agentes **cooperativos**. Los primeros son también conocidos como agentes *egoístas* que buscan conseguir sus objetivos en el sistema compitiendo con otros agentes y normalmente actúan por sí solos. En cambio, los agentes cooperativos colaboran con otros agentes compartiendo recursos y normalmente forman parte de una organización dentro del SMA en la que comparten un fin común.

Centrándonos en la dinámica del rol que los agentes toman en la organización podemos hacer una segunda clasificación de agentes, ya sean individualistas o cooperativos. La mayoría de ellos toman un rol *estático*, realizando un conjunto de tareas fijas y cuyas relaciones con el resto de agentes de la organización son inmutables (e.g., relaciones jerárquicas, cliente-servidor, etc.). Existen otros agentes con un rol *flexible* o *evolutivo*, los cuales pueden cambiar de objetivos, y en consecuencia cambiar el tipo de relaciones con otros agentes (e.g., agentes que pueden tomar varios roles en una organización, agentes que actúan como ofertantes y contratistas a la vez, etc.).

c) Utilidad del agente

Este criterio permite clasificar agentes según el propósito con el que fueron creados. Los factores que determinan la utilidad de un agente vienen dados por el dominio de aplicación y por el tipo de tareas que se va a realizar en tal dominio. Veamos estos factores.

En primer lugar, las áreas donde se ha aplicado el paradigma de agente software han ido creciendo progresivamente: comercio electrónico, telecomunicaciones, gestión de procesos de administración, gestión de conocimiento, etc. En segundo lugar, las tareas de los agentes en estos dominios van desde la monitorización de sistemas hasta la elaboración de diagnósticos, pasando por la búsqueda de información, tareas de mediación y tareas de coordinación y resolución de conflictos entre agentes.

B.3. Sistemas Multiagente

Como hemos visto en la Sección 4.3.1, la idea original del concepto de agente está basada en representar entidades autónomas de forma que éstas puedan interaccionar entre ellas para la gestión inteligente de problemas complejos que no pueden ser abordados individualmente. Esta idea ha dado lugar a la construcción de sistemas multiagente (SMA, o *multiagent systems* (MAS) en inglés) [160,256,261]. El objetivo de estos sistemas es coordinar los distintos agentes que lo componen e integrar sus objetivos particulares en un objetivo común. Las aplicaciones de SMAs pueden abarcar diversos casos: la gestión de problemas cuyos elementos están distribuidos físicamente, cuando la solución para resolver un problema consiste en emplear un conocimiento heterogéneo mantenido por diferentes agentes, cuando cada agente ofrece una solución distinta al mismo problema y es necesario comparar y coordinar dichas soluciones para encontrar cuál es la mejor, etc.

Los sistema multiagente ofrecen un método par evitar las situaciones problemáticas

descritas. En un sistema multiagente están activos diversos agentes autónomos independientes [41]. Cada uno de estos agentes se dedica a sus propios objetivos y sólo contacta con los otros agentes para obtener información, o para contribuir a una solución coordinada de un problema general. En ambas situaciones, cada agente individual tiene una tarea específica para la cual es adecuado y cuya solución no excede de sus capacidades. Esto permite el procesamiento de problemas complejos.

Los sistemas multiagente proporcionan una gran ventaja: permiten la integración de agentes existentes en un gran sistema. Por tanto, la solución de un problema no requiere el diseño y desarrollo de un nuevo agente especializado, en su lugar, puede ser utilizado el conocimiento de los agentes existentes combinándolo en un sistema multiagente o permitiendo que trabajen conjuntamente para resolver el problema.

B.4. Comunicación en Sistemas Multiagente

El problema que surge cuando dos seres inteligentes intentan interactuar es que necesita unas normas y un canal de comunicación: deben utilizar el mismo lenguaje, estar de acuerdo en el significado de los símbolos de ese lenguaje, tener un mecanismo de comunicación para el intercambio de mensajes, no hablar al mismo tiempo, etc. En resumen, debe de existir comunicación y coordinación entre ellos.

B.4.1. Coordinación entre agentes

Cuando el número de agentes del sistema crece, aparece la necesidad de ayuda para localizar otros agentes que tengan la información o que nos puedan ofrecer los servicios que requerimos [207]. Muchos sistema multiagente utilizan agentes mediadores o facilitadotes que proporcionan determinados servicios y asistencia a los agentes del sistema (matchmaker, broker, blackboard, etc.). En general podemos distinguir tres categorías de agentes: P-agentes o agentes proveedores, R-agentes o agentes requeridores y agentes mediadores o intermediarios.

La comunicación forma la base de la coordinación y está formada por los protocolos de comunicación y el resultado de los métodos de comunicación.

B.4.2. Métodos de comunicación

Existen diferentes métodos de comunicación. La invocación del procedimiento de un agente por otro agente representa el caso más simple. No obstante, sólo métodos de comunicación simples pueden implementarse utilizando llamadas a procedimientos. Por esta razón, las

llamadas a procedimientos no se consideran un método de comunicación.

Los métodos de comunicación se pueden diferenciar en sistemas de pizarra y sistemas de mensaje/diálogo. Debido a que sus capacidades se ajustan mejor a los requerimientos de este trabajo nos centraremos en los primeros.

B.4.2.1. Sistemas de pizarra

La arquitectura de pizarra es ampliamente utilizada en Inteligencia Artificial. En un sistema con múltiples fuentes de conocimiento (o agentes independientes) necesitamos un mecanismo de comunicación. La pizarra ("blackboard") es una estructura de datos que es usada como un mecanismo general de comunicación entre las múltiples fuentes de conocimiento y es gestionada y arbitrada por un controlador [99,202,203]. Así, será un área de trabajo común a los agentes donde poder intercambiar información, datos y conocimiento.

El paradigma de pizarra se presenta a menudo usando la siguiente metáfora [257]:

> Imagine un grupo de especialistas, humanos o agentes, sentados cerca de una gran pizarra. Los especialistas están trabajando cooperativamente para resolver un problema, usando la pizarra como el lugar de trabajo en el que desarrollar la solución. La resolución del problema comienza cuando el problema y los datos iniciales son escritos en la pizarra. Los especialistas miran la pizarra, buscando un oportunidad para aplicar su experiencia en la resolución. Cuando un especialista encuentra suficiente información para realizar una contribución, escribe esta en la pizarra. Esta información adicional puede permitir a otros especialistas aplicar su experiencia. Este proceso de añadir contribuciones a la pizarra continua hasta que el problema se ha resuelto.

Esta metáfora captura alguna de las características más importantes de los sistemas de pizarra, que se describen a continuación.

Independencia de experiencia. Los especialistas no están entrenados solamente para trabajar con este grupo específico de especialistas. Cada uno es un experto en algunos aspectos del problema y puede contribuir a la solución independientemente del conjunto de especialistas que participen.

Diversidad en técnicas de resolución de problemas. En los sistemas de pizarra, la representación interna y la maquinaria de inferencia usada por cada especialista están ocultos a la vista.

Representación flexible de la información de la pizarra. El modelo de pizarra no impone restricciones acerca de que información puede incluirse en la misma.

Lenguaje de interacción común. Los especialista en los sistemas de pizarra deben

ser capaces de interpretar correctamente la información guardada en la pizarra por otros especialistas. En la práctica, hay una solución de compromiso entre la expresividad de una representación especializada compartida por unos pocos especialistas y una representación general entendida por todos.

Necesidad de control. Un componente de control distinto de los especialistas es responsable de dirigir el curso de la resolución del problema. El componente de control se puede ver como un especialista en la dirección de resolución de problemas.

Generación incremental de la solución. Los especialistas contribuyen a la solución como sea más apropiado, algunas veces refinando, otras contradiciendo y otras iniciando una nueva línea de razonamiento.

Como cada agente trabaja con su parte del problema, acudirá a la pizarra para ver nueva información puesta por otros agentes y, a su vez, pondrá sus resultados. La forma de funcionar será: un agente inicia una comunicación escribiendo algún tipo de información en la pizarra; en ese momento, estos datos están disponibles para todos los agentes del sistema; cada agente accederá a la pizarra eventualmente para comprobar si hay información nueva; normalmente cada agente no recogerá toda la información que se vaya escribiendo en la pizarra, sino que sólo obtendrá aquella que le interese, tal vez por pertenecer a conocimiento afín. Como se puede observar no hay comunicación directa entre agentes. La figura B.3 muestra la estructura de un sistema de pizarra.

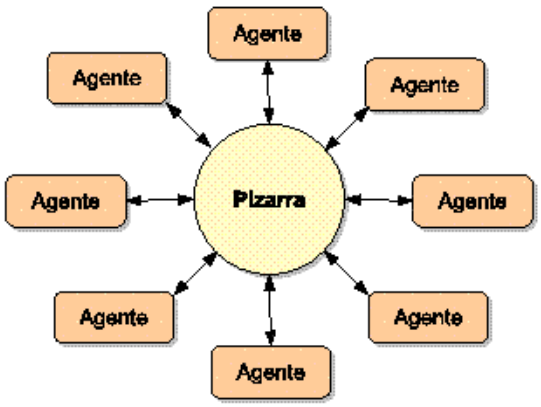


Figura B.3: Estructura de un sistema de pizarra

En ciertos casos se ha introducido la idea del moderador, que se encarga de supervisar el estado del problema, los siguientes subprobemas a ser resueltos y además trata de organizar el trabajo entre y hacia los agentes. Cualquier agente puede usar la pizarra para leer los subproblemas generados. El moderador controla y evalúa el conocimiento de la base de conocimiento para seleccionar a los agentes más capacitados para resolver el subproblema.

Otra mejora es incluir un dispatcher que se encarga de informar a los agentes interesados de nueva información que se escribe en la pizarra, en vez de que éstos accedan regularmente a ella.

Los sistemas de pizarra ofrecen un sistema flexible para la cooperación y comunicación entre las unidades que componen un sistema distribuido, ya que son independientes de las estrategias de cooperación.

B.4.2.2. Sistemas de mensajes

Para desarrollar este tipo de interacción entre agentes existen tres elementos fundamentales a tener en cuenta. El primero es un *lenguaje de comunicación* para definir la estructura de los mensajes y sus actos comunicativos asociados, i.e. las acciones que representan (e.g., una orden, una propuesta, un paso de información, etc.). El segundo es una tecnología para representar el *contenido* del mensaje, i.e. la proposición que acompaña a la acción (e.g., la tarea a realizar o una pieza de información concreta). El tercer elemento es el *protocolo de interacción*, i.e. el conjunto de patrones de secuencia de mensajes que permite construir conversaciones entre agentes. En el resto de esta sección veremos cada uno de estos elementos con más detalle.

Lenguajes de comunicación

Los mensajes intercambiados entre agentes deben ser dotados de una estructura y del tipo de acción que representan para que puedan ser correctamente procesados. Los lenguajes de comunicación declarativos [114,171] son el enfoque más aceptado y utilizado para este fin. Por un lado, estos lenguajes definen una serie de elementos básicos que componen la estructura de un mensaje (e.g., identificadores de los agentes emisor y receptor, tipo de acto comunicativo, contenido, etc.). Por otro lado, permiten indicar el tipo de acción que representa un mensaje siguiendo la teoría de los actos del habla (*speech acts)* de Searle [239].

Según la teoría mencionada, cualquier acto comunicativo puede ser modelado mediante declaraciones denominadas *performativas*. Dichas declaraciones no están orientadas a describir o constatar un hecho u opinión, sino que su objetivo es realizar una acción para producir un determinado efecto en el receptor. Ejemplos de performativas son acciones comunicativas que representan órdenes o peticiones hechas a un agente ("Quiero que ejecutes la tarea T"), afirmaciones ("Te informo de que mañana va a llover"), etc.

Normalmente, las performativas se agrupan en diferentes clases según la actividad a la que se encuentran destinadas. Las tres clases básicas que aparecen en los lenguajes de comunicación declarativos son las performativas de información, de realización y de negociación. Las performativas de información (e.g., *inform*, *query*, etc.) permiten informar y preguntar sobre

la veracidad o falsedad de una proposición. Las performativas de realización (e.g., *request*, *agree*, *refuse*, etc.) permiten dar órdenes para llevar a cabo ciertas tareas y aceptar o no tales órdenes. Las performativas de negociación (e.g., *propose*) se utilizan para iniciar un proceso de negociación general o proponer una acción particular.

Es importante resaltar que los primeros SMAs usaban sus propios lenguajes de comunicación para resolver sus necesidades comunicativas. Obviamente, agentes en diferentes SMAs podían no usar el mismo lenguaje provocando la falta de interoperabilidad entre ellos. Como solución se propuso crear estándares para definir lenguajes de comunicación. El lenguaje estándar más utilizado actualmente es ACL (*Agent Communication Language*) [115], propuesto por FIPA (*Foundation for Intelligent Physical Agents*) [1] . Este lenguaje define una estructura extensible de mensajes y contiene varias librerías de performativas descritas mediante un modelo formal, acompañadas de explicaciones detalladas en lenguaje natural y ejemplos de uso.

Tecnologías para representar el contenido de los mensajes

El contenido de un mensaje equivale a las proposiciones intercambiadas entre agentes. Dependiendo de la performativa del mensaje, estas proposiciones pueden ser acciones concretas a realizar, sentencias sobre elementos del dominio, etc. A la hora de describir este contenido hay que tener en cuenta que la representación utilizada tiene que ser compartida por todos los agentes de la misma manera. Es decir, el hecho de que dos agentes interaccionen usando el mismo lenguaje comunicativo, como por ejemplo ACL, no asegura que puedan llegar a entenderse si no comparten el mismo significado para el contenido del mensaje.

Para abordar el problema de una representación común para el contenido de los mensajes surge la idea de *compartición de conocimiento*. Esta idea reside en expresar la información del dominio sobre el que están trabajando los agentes mediante *ontologías*. Básicamente, una ontología permite definir formalmente los términos y relaciones que componen el vocabulario de un dominio junto con una serie de operadores para combinar ambos elementos. Una vez definido, el vocabulario es compartido por todos los agentes del sistema y usado por éstos para realizar sentencias concretas sobre el dominio.

Protocolos de interacción

Los protocolos de interacción consisten en un conjunto de patrones compartidos entre agentes para controlar el intercambio de mensajes. El fundamento de estos protocolos es definir ciertas secuencias de mensajes para que los agentes involucrados en una conversación

[1] FIPA está disponible en `http://www.fipa.org/`

alcancen sus objetivos de una forma eficiente.

Para definir un protocolo de interacción hay que determinar el siguiente conjunto de elementos: el rol de los agentes en la conversación (e.g., iniciador, ofertante, contratante, etc.), el conjunto de performativas que se pueden utilizar, el conjunto de mensajes permitido para responder un mensaje recibido, los mensajes de inicio de la conversación y los mensajes que causan el fin de ésta, y el resultado obtenido de la conversación para los agentes involucrados.

El conjunto de recomendaciones de FIPA ha definido un lenguaje estándar de representación de protocolos llamado AUML [21]. Este lenguaje está basado en diagramas de secuencia de UML y añade nuevos elementos para poder describir diagramas de protocolo. Ejemplos de protocolos de interacción ya predefinidos en las recomendaciones de FIPA, tales como subastas (*auctions*) o redes de contratos (*contract net*), pueden consultarse en la librería de protocolos de FIPA [116]. Además, este conjunto de protocolos puede expandirse según las necesidades de nuevas conversaciones entre agentes de una forma sencilla siguiendo el lenguaje estándar dado por FIPA.

Apéndice C

Extracción inteligente de conocimiento a partir de datos

C.1. Introducción

La extracción inteligente de conocimiento a partir de datos es un proceso especialmente diseñado para extraer conocimiento útil a partir de los datos históricos almacenados. En particular, con respecto a los objetivos de este trabajo, se considera que será de gran ayuda a la hora de obtener conocimiento interesante a partir de las ejecuciones de distintas metaheurísticas individuales. Este conocimiento servirá para guiar a las metaheurísticas cooperativas diseñadas utilizando marco general propuesto en esta tesis. En este apéndice se hace un estudio de las distintas fases del proceso de extracción inteligente de conocimiento a partir de datos para, en base a esto, definir un proceso de extracción específico para el marco general propuesto. Una vez realizado este estudio nos centramos en la fase más importante del mismo, la minería de datos, realizando un estudio de las distintos modelos y técnicas que se podrían utilizar, con la intención de decidir cual es la más apropiada.

C.2. El proceso de extracción inteligente de conocimiento a partir de datos

Las bases de datos actuales han acumulado una gran variedad y cantidad de datos, estadísticas, índices, etc. en los cuales la información útil no es fácil de encontrar o inferir a simple vista. Con semejante cantidad de datos (y que cada vez crece más), el descubrimiento de conocimiento en bases de datos es una necesidad.

El proceso de extracción inteligente de conocimiento a partir de datos, también conocido como descubrimiento de conocimiento en bases de datos o KDD, se puede definir como el

proceso no trivial de identificar patrones válidos, novedosos, potencialmente útiles y en última instancia, comprensibles a partir de grandes cantidades de datos que pueden estar en diferentes formatos. En esta definición se pueden ver los dos retos de este proceso: por un lado, trabajar con grandes volúmenes de datos, con los problemas que esto puede conllevar (ruido, datos ausentes, intratabilidad, volatilidad de los datos...), y por el otro utilizar técnicas adecuadas para analizar los mismos y extraer conocimiento novedoso y útil.

Como se deduce de la definición anterior, la extracción de conocimiento inteligente es un proceso complejo que incluye no sólo la obtención de modelos y patrones, sino también la evaluación y posible interpretación de los mismos.

De igual modo las aplicaciones de la extracción inteligente de conocimiento a partir de datos se extienden a campos tan variopintos como: Marketing, inversiones, detección de fraude, fabricación, telecomuniciaciones, agricultura, diseño Web y un largo etcétera.

C.3. Las fases del proceso de extracción de conocimiento

El proceso de extracción inteligente de conocimiento, Figura C.1, es un proceso iterativo e interactivo. Es iterativo ya que la salida de alguna de las fases puede hacer volver a pasos anteriores y porque a menudo son necesarias varias iteraciones para extraer conocimiento de alta calidad. Es interactivo, porque el usuario, o un experto en el dominio del problema, debe ayudar en la preparación de datos, validación del conocimiento extraído, etc.

Este proceso se organiza en torno a cinco fases, como se ilustra en la figura C.1, que se explicarán con mayor detenimiento en las siguientes subsecciones.

C.3.1. Fase de integración y recopilación

Cuando se quiere aplicar un proceso de extracción de conocimiento es normal que los datos pertenezcan a diferentes organizaciones o a distintos departamentos de una misma entidad. Incluso en muchos casos se tendrán que adquirir además datos externos desde bases de datos públicas o privadas. Esto representa un reto, ya que cada fuente de datos usa diferentes formatos de registro, diferentes grados de agregación de los datos, diferentes tipos de error, etc. Lo primero, por lo tanto es integrar todos estos datos. La idea de la integración de múltiples bases de datos ha dado lugar a la tecnología de almacenes de datos. Un almacén de datos es un repositorio de información coleccionada desde varias fuentes, almacenada bajo un esquema unificado que normalmente reside en un único emplazamiento. Existen varias formas de mezclar las distintas bases de datos para crear el repositorio, siendo la más utilizada la definición de un proceso de integración y almacenamiento en un nuevo esquema integrado.

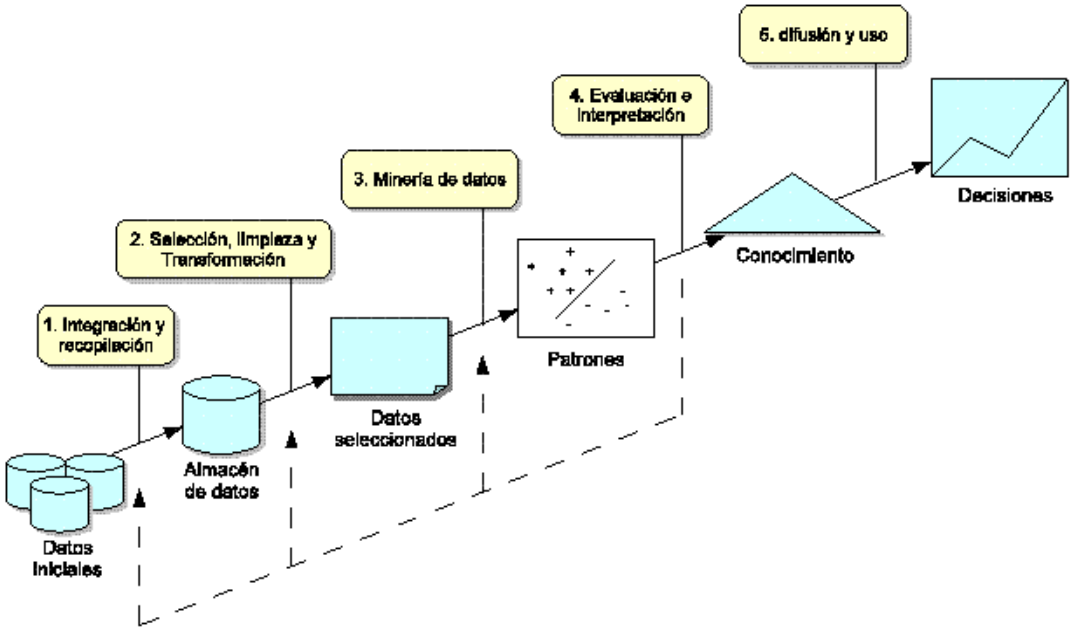


Figura C.1: Fases del proceso de extracción de conocimiento.

C.3.2. Fase de selección, limpieza y transformación

La calidad del conocimiento descubierto no sólo depende del algoritmo de minería utilizado, sino también de la calidad de los datos minados. Por ello, después de la recopilación, el siguiente paso en el proceso de extracción de conocimiento es seleccionar y preparar el subconjunto de datos que se va a minar, los cuales constituyen lo que se conoce como *vista minable*. Este paso es necesario ya que algunos datos coleccionados en la etapa anterior pueden ser irrelevantes o incluso erróneos, y también puede ocurrir que aparezcan datos faltantes.

Estos problemas son sólo algunos ejemplos que muestran la necesidad de la limpieza de datos, es decir, de mejorar su calidad. Pero no es suficiente con tener datos de buena calidad, sino además poder proporcionar a los métodos de minería de datos el subconjunto de datos más adecuado para resolver el problema. Para ello es necesario seleccionar los datos apropiados.

La selección de atributos relevantes es uno de los preprocesamientos más importantes, ya que es crucial que los atributos utilizados sean relevantes para la tarea de minería de datos. Idealmente, se podrían usar todas las variables, pero es más interesante eliminar aquellos atributos que no sean relevantes. De esta forma existen técnicas que permiten obtener las variables más relevantes, aunque nuestro conocimiento sobre el dominio del problema puede permitirnos hacer correctamente muchas de esas selecciones.

Al igual que con las variables, también podríamos construir el modelo usando todos

los datos. Pero si tenemos muchos, tardaríamos mucho tiempo y probablemente también necesitaríamos una máquina más potente. Consecuentemente, una buena idea es usar una muestra a partir de algunos datos, debiendo hacerse cuidadosamente esta selección.

De igual manera, otra tarea de preparación de los datos es la construcción de atributos, que consiste en construir nuevos atributos aplicando alguna operación o función a los atributos originales con objeto de hacer más fácil el proceso de minería.

Por último, el tipo de los datos puede también modificarse para facilitar el uso de técnicas que requieren tipos de datos específicos. Así, algunos atributos se pueden numerizar, lo que reduce el espacio y permite usar técnicas numéricas. Y otros se pueden discretizar, es decir, transformar los valores numéricos en atributos discretos o nominales.

C.3.3. Fase de minería de datos

La fase de minería de datos es la más característica de la extracción inteligente de conocimiento y, por esta razón, muchas veces se utiliza esta fase para nombrar todo el proceso. El objetivo de esta fase es producir nuevo conocimiento que pueda utilizar el usuario. Esto se realiza construyendo un modelo basado en los datos recopilados para este efecto. El modelo es una descripción de los patrones y relaciones entre los datos que pueden usarse para hacer predicciones, para entender mejor los datos o para explicar situaciones pasadas. Para ello es necesario tomar una serie de decisiones antes de empezar el proceso:

- Determinar le tipo de tarea de minería más apropiado. Por ejemplo, podríamos usar la clasificación para predecir en una entidad bancaria los clientes que dejarán de serlo.
- Elegir el tipo de modelo. Por ejemplo, para una tarea de clasificación podríamos usar un árbol de decisión, porque queremos obtener un modelo en forma de reglas.
- Elegir el algoritmo de minería que resuelva la tarea y obtenga el tipo de modelo que estamos buscando. Esta elección es pertinente porque existen muchos métodos para construir los modelos.

Es en la construcción del modelo donde vemos mejor el carácter iterativo del proceso de extracción de conocimiento, ya que será necesario explorar modelos alternativos hasta encontrar aquel que resulte más útil para resolver nuestro problema. Así, una vez obtenido un modelo y a partir de los resultados obtenidos para el mismo, podríamos querer construir otro modelo usando la misma técnica pero otros parámetros, o quizás usar otras técnicas o herramientas. En esta búsqueda del "buen modelo" puede que tengamos que retroceder hasta fases anteriores y hacer cambios en los datos que estamos usando o incluso modificar la

definición del problema. A continuación se explicarán las tareas de la minería de datos, y en la siguiente sección se hará un estudio de los modelos y técnicas.

C.3.3.1. Tareas de la minería de datos

Dentro de la minería de datos hemos de distinguir tipos de tareas, cada una de las cuales puede considerarse como un tipo de problema a ser resuelto por un algoritmo de minería de datos. Cada tarea tiene sus propios requisitos, y el tipo de información obtenida puede diferir mucho dependiendo de la misma.

Las distintas tareas pueden ser predictivas o descriptivas. Entre las tareas predictivas encontramos la clasificación y la regresión, mientras que el agrupamiento (clustering), las reglas de asociación, las reglas de asociación secuenciales y las correlaciones son tareas descriptivas. Veamos en mayor detalle todas ellas.

La clasificación es quizá la tarea más utilizada. En ella, cada instancia, o ejemplo, pertenece a una clase, la cual se indica mediante el valor de un atributo que llamamos la clase de la instancia. El objetivo es predecir la clase de nuevas instancias de las que se desconoce la clase.

La regresión es también una tarea predictiva que consiste en aprender una función real que asigna a cada instancia un valor real.

El **agrupamiento** es la tarea descriptiva por excelencia y consiste en obtener grupos "naturales" a partir de los datos. A diferencia de la clasificación, en lugar de analizar datos etiquetados con una clase, los analiza para generar esta etiqueta. Los datos son agrupados basándose en el principio de maximizar la similitud entre los elementos de un grupo minimizando la similitud entre los distintos grupos.

Las correlaciones son una tarea descriptiva que se usa para examinar el grado de similitud de los valores de dos variables numéricas.

Las reglas de asociación son también una tarea descriptiva, muy similar a las correlaciones, que tiene como objetivo identificar relaciones no explícitas entre atributos *categóricos*

Un caso especial de las reglas de asociación son las **reglas de asociación secuenciales**, que se usan para determinar patrones secuenciales en los datos.

C.3.4. Fase de evaluación e interpretación

Medir la calidad de los patrones descubiertos por un algoritmo de minería de datos no es un problema trivial, ya que esta medida puede atañer a varios criterios, algunos de ellos bastante subjetivos. Idealmente, los patrones descubiertos deben tener tres cualidades: ser

precisos, comprensibles e interesantes. Según las aplicaciones puede interesar mejorar algún criterio y sacrificar ligeramente otro, como en el caso del diagnóstico médico que prefiere patrones comprensibles aunque su precisión no sea muy buena.

Existen distintas técnicas de evaluación como son, la validación simple, la validación cruzada o el bootstrapping, para una descripción más detallada consultar [146].

Dependiendo de la tarea de minería de datos existen diferentes medidas de evaluación de los modelos. En el contexto de la clasificación, lo normal es utilizar la precisión predictiva. En el caso de las reglas de asociación, se suelen utilizar medidas como la cobertura y la confianza. En regresión, se utiliza el error cuadrático medio y en agrupamiento suele evaluarse en función de la cohesión de cada grupo y de la separación entre grupos.

C.3.5. Fase de difusión, uso y monitorización

Una vez construido y validado, el modelo puede usarse principalmente con dos finalidades: para que un analista recomiende acciones basándose en el modelo y en sus resultados, o bien para aplicar el modelo a diferentes conjuntos de datos. Tanto de una forma como de otra es necesaria la difusión del modelo, es decir, que se distribuya y se comunique a los posibles usuarios. Además, el modelo deberá ser monitorizado, lo que significa que, periódicamente, el modelo tendrá que ser re-evaluado, re-entrenado y posiblemente reconstruido completamente.

C.4. Técnicas de Minería de datos

En esta sección se realiza una revisión y una clasificación no exhaustiva de las técnicas disponibles en la fase de minería de datos. Esta clasificación se realiza desde distintos enfoques, en concreto, a) según el tipo de modelo generado, b) según el tratamiento de la imperfección en los datos que permite la técnica y por último, c) en función del tipo de dependencia capturada por el modelo, ya sea global o parcial. El objetivo de esta revisión y clasificación es facilitar la elección de las técnicas que se utilizarán en el proceso de extracción inteligente de conocimiento definido en el Capítulo 5.

C.4.1. Técnicas y Modelos

El objetivo de la fase de minería de datos es producir nuevo conocimiento que pueda utilizar el usuario. Esto se realiza construyendo un modelo basado en los datos recopilados para este efecto. El modelo es una descripción de los patrones y relaciones entre los datos que pueden usarse para hacer predicciones, para entender mejor los datos o para explicar

situaciones pasadas.

Existen muchas formas de representar los modelos y cada una de ellas determina el tipo de técnica que puede usarse para inferirlos. Como ya se ha comentado, los modelos pueden ser de dos tipos: predictivos y descriptivos.

Existen múltiples técnicas de minería de datos que podemos utilizar y en este apartado las agruparemos en función del tipo de modelo obtenido. Existen diferentes paradigmas detrás de las técnicas utilizadas atendiendo a la forma de los modelos que obtienen, sin ser exhaustivos, encontramos: funciones discriminantes estadísticas y no estadísticas, redes bayesianas, árboles de decisión, redes neuronales, modelos basados en reglas, modelos basados en patrones muestrales, ...

- Una de las formas más útiles de representar un clasificador es mediante un conjunto de funciones discriminantes. El clasificador en este caso puede ser visto como una máquina que computa c funciones discriminantes $g_i(x), i = 1, ..., c$, una para cada clase, y selecciona para x la clase con mayor valor para la función discriminante [105]. De esta forma puede ser expresado el clasificador de Bayes donde $g_i(x) = P(\omega_i/x)$, de forma que la máxima función discriminante es la máxima probabilidad a posteriori. De las funciones de densidad que pueden determinar el clasificador bayesiano, la más investigada ha sido la densidad Gaussiana, debido a su posible tratamiento analítico y a que sirve para modelar de forma apropiada una gran variedad de situaciones. Cuando suponemos que la forma de la función de densidad es conocida, la estimación de los parámetros de estas funciones a partir de ejemplos se denomina estimación paramétrica y se puede realizar mediante el método de máxima verosimilitud que trata de obtener el conjunto de parámetros que maximiza la verosimilitud de los ejemplos de la muestra o mediante estimación bayesiana, que considera al conjunto de parámetros como otra variable del problema y a diferencia con el método de la máxima verosimilitud tiene en cuenta, no sólo la información extraída de la muestra, sino toda la información disponible de los parámetros de la función de densidad. Cuando las funciones discriminantes son funciones de densidad complejas, éstas pueden ser aproximadas por una mezcla de funciones de densidad más simples. De nuevo, los parámetros de las funciones de densidad componentes pueden ser obtenidos mediante estimación de la máxima verosimilitud o estimación bayesiana. Sin embargo, cuando hacemos la estimación de la máxima verosimilitud, la solución no se puede encontrar analíticamente y la mayoría de técnicas empleadas para obtener una solución, parten de una inicialización aleatoria de los parámetros y mediante métodos iterativos (como el conocido algoritmo EM) se van modificando sus estimaciones [95, 187]. Cuando las densidades componentes son

densidades gaussianas, se obtienen algunas simplificaciones asumiendo que las matrices de covarianzas son diagonales, que son iguales o que los componentes agrupan patrones muestrales con independencia en sus atributos [47, 235]. Se puede generalizar este tipo de funciones discriminantes, mediante el uso de teorías que generalizan el modelo bayesiano como la de probabilidades imprecisas, que permiten probabilidades inferiores y superiores, previsiones inferiores y superiores [254].

Otro tipo importante de funciones discriminantes son las funciones discriminantes lineales o funciones discriminantes que son una combinación lineal de los componentes del vector de entrada x, $g_i(x) = w_i x + w_{i0}; i = 1, ..., c$. Estas funciones discriminantes dividen el espacio de características mediante hiperplanos donde la orientación del hiperplano la fija el vector w y la localización el valor w_{i0}, dando lugar a c regiones de decisión y $g_i(x)$ será la función de máximo valor si x cae en la región R_i. Hay distintos procedimientos para obtener el conjunto de pesos de estas funciones discriminantes como gradiente descendente, el algoritmo de Newton, criterio del Perceptrón [118, 165]. En este último trabajo se propone una versión difusa del algoritmo del Perceptrón. Cuando se busca el hiperplano separador que está a la misma distancia de los ejemplos más cercanos de cada clase, hablamos de las máquinas de vectores soporte [249]. El aprendizaje de las máquinas de vectores soporte es un problema de optimización con restricciones que se puede resolver usando técnicas de programación cuadrática. El aprendizaje de separadores no lineales mediante máquinas de vectores soporte se consigue mediante una transformación no lineal del espacio de atributos de entrada en un espacio de características de dimensionalidad mucho mayor y donde sí es posible separar linealmente los ejemplos. El uso de funciones núcleo permite trabajar eficientemente en el espacio de características sin necesidad de calcular explícitamente las transformaciones en los ejemplos de aprendizaje [237].

- Como hemos comentado anteriormente, las técnicas bayesianas calculan, para una instancia dada sin clasificar, cuál es la probabilidad de que se le asigne cada una de las clases, y selecciona la de mayor probabilidad. La teoría de la probabilidad modela la incertidumbre asignando una probabilidad a cada uno de los estados del mundo. Cada uno de estos estados es un elemento del conjunto de posibles configuraciones de los atributos que lo rigen. Un modelo probabilístico determina una probabilidad conjunta sobre todos los valores de los atributos del modelo. El problema está en que el número de valores que determinan la distribución conjunta crece de manera exponencial en el número de variables. Las redes Bayesianas [119] modelan el sistema bajo estudio mediante una estructura que permite suavizar este problema. La clave está en mostrar

una estructura que es más real, la influencia local de las variables: cada variable realmente está influenciada directamente por unas pocas variables. Más concretamente, la semántica de las redes Bayesianas se basa en el concepto de dependencia condicionada. Se pueden aprender las relaciones de dependencia e independencia entre todas las variables que conforman el dominio de estudio, permitiendo así realizar predicciones sobre el comportamiento de cualquiera de las variables desconocidas a partir de los valores de las otras variables conocidas. Las redes Bayesianas pueden realizar la tarea de clasificación como un caso particular de la tarea de predicción mencionada anteriormente.

- Las técnicas basadas en patrones muestrales aproximan una función de densidad desconocida mediante una versión promediada de dicha densidad a través de la probabilidad de que un determinado vector caiga en una región concreta dentro del espacio de características [105]. Estas técnicas carecen de fase de aprendizaje ya que el modelo del problema está formado por el conjunto de patrones muestrales de partida. Hay dos formas comunes de especificar dichas regiones: especificando un volumen como función del número de ejemplos (método de Parzen) o especificando el número de ejemplos que forman parte de dicha región y por lo tanto el volumen aumenta hasta que englobe dicho número de ejemplos (método de k vecinos). Se han propuesto versiones difusas de esta regla en [26, 164], permitiendo gradaciones en la pertenencia de un ejemplo a cada una de las clases.

- Las técnicas que modelan el sistema mediante árboles de decisión son útiles para encontrar estructuras en espacios de alta dimensionalidad o cuando las densidades condicionales de las clases son desconocidas o son multimodales. Los árboles de decisión, son una serie de decisiones organizadas en forma jerárquica, a modo de árbol. Un árbol de decisión es una estructura de conocimiento que permite asignar un valor para el atributo de salida de un objeto dado, haciendo preguntas sobre los valores de los restantes atributos del objeto. Existen multitud de algoritmos para la construcción de árboles de decisión que se diferencian fundamentalmente en las particiones que consideran y el criterio para la selección de una partición. Uno de los algoritmos más utilizados dentro de este enfoque es el ID3 [223]. Se trata de un algoritmo capaz de generar un árbol de decisión óptimo considerando únicamente atributos nominales. Una generalización de este algoritmo es el popular C4.5 [224,225] que permite trabajar con atributos nominales y numéricos. Mientras que estos dos algoritmos generan árboles de clasificación, CART es un algoritmo que genera árboles de regresión [40]. Se han propuesto versiones difusas de estos algoritmos como [159].

- Los árboles de decisión pueden considerarse una forma de aprendizaje de reglas (difusas o no), ya que cada rama del árbol puede interpretarse como una regla. Las técnicas basadas en reglas modelan el sistema mediante una base de reglas construidas a partir de una muestra de ejemplos. De manera genérica podemos decir que la estructura de dichas reglas es de la forma "si <antecedente o condición> entonces <consecuente o conclusión>". El <antecedente> y el <consecuente> (que pueden ser difusos) son proposiciones que pueden formarse usando conjunciones o disyunciones. Una vez construida la base de reglas, el proceso asigna una salida a un ejemplo desconocido con un cierto grado (con grado 0 ó 1 en el caso clásico, o con cierto grado entre 0 y 1 en el caso difuso). Algunos métodos de obtención de reglas (reglas de asociación) se basan en el concepto de conjuntos de ítems frecuentes y utilizan técnicas de conteo y soporte mínimo [178]. Otros métodos añaden reglas mientras vayan cubriendo ejemplos. Éstos son los llamados métodos por cobertura como los basados en los algoritmos AQ [191] y CN2 [72]. Cuando en las reglas se permiten comparaciones entre atributos se obtienen predicados más complejos que dan lugar a las reglas generadas por las técnicas de Programación Lógica Inductiva [107]. Los algoritmos genéticos [177] también se han utilizado como técnicas de generación de reglas. Existe otro grupo de técnicas con el objetivo común de generar agrupaciones en los datos. Éstas son las técnicas denominadas de agrupamiento o clustering. El análisis cluster es el estudio formal de los algoritmos y métodos de agrupamiento o clasificación de ejemplos. El análisis cluster no utiliza etiquetas de clase que señalen una identificación previa de los ejemplos, sino que el objetivo del análisis cluster es simplemente el encontrar una organización válida y conveniente para los datos e intentar encontrar su estructura subyacente. Al existir agrupamientos en los datos utilizados para el aprendizaje, podremos crear patrones característicos de comportamientos, o lo que es igual, ser capaces de asociar a cada grupo de datos un elemento característico. Estos elementos característicos pueden ser interpretados como reglas cuyo consecuente estará formado por el atributo o atributos a estimar y el antecedente por los restantes atributos conocidos del ejemplo. Dentro de estas técnicas de agrupamiento podemos situar los mapas auto-organizativos de Kohonen [166], las basadas en el algoritmo de las K medias que se encuentran dentro de lo que se conoce agrupamiento por partición [196], en contraposición de los métodos jerárquicos que no establecen a priori el número de grupos a generar [117]. También las técnicas que modelan mediante una mezcla de densidades de probabilidad pueden ser consideradas como técnicas de agrupamiento donde cada grupo es cada componente de la mezcla y por tanto cada componente una regla [47, 235]. Dentro de este grupo de técnicas que generan un conjunto de reglas cabe destacar que una vez generadas las reglas se suelen

emplear técnicas que permitan ajustar esas reglas (reducir el número de reglas, modificar dichas reglas, ...). Ésto se realiza con multitud de técnicas que recorren desde el clásico método de minimizar el error cuadrático medio hasta el uso de redes neuronales [201] y algoritmos genéticos [80].

- Otro tipo de modelo generado por diversas técnicas de minería de datos es el de red neuronal. Una red neuronal puede ser considerada como un grafo dirigido con muchos nodos (elementos de proceso) y arcos entre ellos (sus interconexiones). Cada uno de estos elementos funciona independientemente de los demás, usando datos locales (la entrada y la salida del nodo) para dirigir su procesamiento. Las redes neuronales son un paradigma de computación muy potente que permite modelizar problemas complejos en los que puede haber interacciones no lineales entre las variables, aunque requieren bastantes datos para el entrenamiento. Su mayor desventaja es que el modelo obtenido es difícil de comprender. Entre las redes que han adquirido más importancia se encuentra el Perceptrón multicapa que generaliza el algoritmo del Perceptrón con la idea de generar más de una frontera de separación en el espacio de características [118]. Una extensión de esta técnica es la versión difusa propuesta en [194].

En los últimos años ha surgido un interés creciente en la construcción de meta-modelos cuya idea es la de utilizar un conjunto de modelos diferentes para combinarlos creando un nuevo modelo. Así podemos encontrar trabajos en este sentido como los basados en técnicas de Fuzzy Pattern Matching [102]. En particular, estas técnicas han sido exitosamente aplicadas dentro del aprendizaje inductivo para resolver las tareas de clasificación o regresión. Podemos encontrar aplicaciones de fuzzy pattern matching en el campo del meta-aprendizaje, [48], en el que se realiza una fusión de distintos clasificadores.

C.4.2. Técnicas y Tratamiento de la Imperfección

Como hemos visto, existe una amplia variedad de técnicas de minería de datos basadas en diferentes propuestas teóricas [105, 225, 260]. Sin embargo, la mayoría de las técnicas convencionales de construcción de modelos para realizar las tareas propias de la Minería de Datos han prestado relativamente poca atención a las fuentes de incertidumbre y los datos incompletos e imprecisos son descartados o ignorados tanto para el proceso de aprendizaje como para el de inferencia posterior.

Sin embargo, las observaciones imperfectas aparecen de forma inevitable en dominios y situaciones realistas. Los errores instrumentales o corrupción por ruido durante los experimentos pueden dar ocasión a datos incompletos cuando estamos midiendo un atributo específico

para la observación en cuestión. En otros casos, la extracción de la información exacta puede ser en exceso costosa o no viable. Además, en ocasiones puede ser útil usar información adicional de un experto y que por lo general está dada por conceptos difusos: pequeño, más o menos, cerca de, etc.

Veamos en primer lugar qué entendemos por información imperfecta, para pasar posteriormente a clasificar las técnicas de minería de datos en función del tipo de información imperfecta que permiten en el conjunto de datos de entrada.

La imprecisión y la incertidumbre pueden ser considerados como dos aspectos complementarios de información imperfecta [104]. Desde el punto de vista práctico, un item de información puede ser representado como una cuádrupla (atributo, objeto,valor,confianza). El atributo es una función que asigna un valor (o conjunto de valores) al objeto. El valor es un subconjunto del dominio de referencia asociado al atributo. La confianza es una indicación de la veracidad del item de información. En este contexto, la imprecisión está relacionada con el valor del item de información, mientras que la incertidumbre está relacionada con la confianza del item. De esta forma, un item de información será preciso cuando su valor no puede ser subdividido. En otro caso hablaremos de información imprecisa. Además, cuando no hay unos límites claros en el conjunto de valores que toma un item impreciso hablaremos de imprecisión difusa. Por otro lado, la incertidumbre es una propiedad de la creencia. Decimos que tenemos certeza acerca de un suceso si le asignamos el máximo valor de creencia. La incertidumbre la podemos definir como la ausencia de la certeza y podrá provenir de la aleatoriedad en cierta experimentación (incertidumbre objetiva) o de juicios subjetivos en el razonamiento humano (incertidumbre subjetiva).

Por lo tanto sería interesante incorporar el manejo, tanto en las fases de aprendizaje e inferencia de las distintas técnicas, de datos que pudieran estar descritos mediante atributos heterogéneos (numéricos y nominales o simbólicos) y que puedan presentar incertidumbre e imprecisión. Así, hablaremos de atributos expresados mediante valores exactos (crisp), valores desconocidos (missing), valores con incertidumbre objetiva, valores con incertidumbre subjetiva, valores imprecisos no difusos y valores imprecisos difusos o etiquetas lingüísticas.

Algunas técnicas sólo pueden trabajar con atributos de dominio nominal, teniendo que ser discretizado el dominio de los atributos numéricos, como ocurre con las redes bayesianas y técnicas basadas en probabilidades imprecisas. Hay técnicas que están limitadas a trabajar con atributos numéricos y sólo permiten un atributo nominal que es el considerado como clase. Son aquellas técnicas que sólo pueden ser empleadas en clasificación como las técnicas que generan funciones discriminantes o redes neuronales. Las técnicas que permiten trabajar con ambos tipos de atributos son, entre otras, las basadas en la construcción de árboles de

clasificación/regresión, las técnicas basadas en patrones muestrales mediante la definición de funciones distancia heterogéneas y, en general, los métodos que generan reglas ya que permiten que éstas estén expresadas usando ambos tipos de atributos salvo las reglas de asociación que sólo trabajan con dominios discretos.

Se han realizado algunos esfuerzos para incorporar incertidumbre e imprecisión en las fases de aprendizaje e inferencia de técnicas de minería de datos ampliamente conocidas. Así, de manera no exhaustiva, algunas técnicas permiten el tratamiento de ejemplos con algún atributo desconocido como es el caso de las técnicas basadas en la estimación de densidades a partir de patrones muestrales [260], técnicas de construcción de árboles de regresión/clasificación [105, 225], el clasificador basado en funciones discriminantes constituidas por normales multivariantes [187], clustering basado en modelos de mezclas [235], redes bayesianas [146], etc. En las técnicas para la construcción de árboles y en modelos de mezcla también se permiten observaciones con atributos nominales que presentan valores inciertos desde el punto de vista probabilista, así como atributos continuos expresados mediante intervalos clásicos (imprecisión no difusa) [225, 235]. También se incorpora el tratamiento de atributos expresados mediante valores difusos en las técnicas para la construcción de árboles [159], así como en redes neuronales [194] y en modelos de mezclas [47]. En ésta última técnica se permite el tratamiento de valores inciertos expresados mediante probabilidades imprecisas, permitiendo por tanto el tratamiento de incertidumbre subjetiva.

En general, podemos decir que hay bastantes limitaciones acerca de la imperfección permitida en las observaciones de partida. Las técnicas que en la actualidad realizan un tratamiento más completo de información imperfecta son los árboles de clasificación y los modelos de mezcla.

A modo de resumen podemos decir que ambas técnicas permiten tratar con ejemplos expresados con atributos nominales y numéricos con algún atributo desconocido, o dado mediante imprecisión no difusa (intervalo clásico), o con imprecisión difusa (etiqueta lingüística), o mediante una distribución de probabilidad en atributos nominales. Además en el modelo de mezcla también se permiten valores inciertos desde un punto de vista subjetivo y que pueden estar expresados mediante probabilidades imprecisas.

C.4.3. Técnicas y Captura de Dependencias

Cuando intentamos obtener un modelo de las dependencias relevantes entre los atributos conocidos y desconocidos de un ejemplo para realizar tareas de inferencia, dicho modelo puede capturar las dependencias de dos formas:

- Mediante una dependencia global, es decir, entre todos los atributos.

  Esto permite que con una sóla etapa de aprendizaje podamos posteriormente inferir los valores de cualquier subconjunto de atributos a partir de cualquier otro subconjunto de atributos con valores conocidos en dicho ejemplo. Esto es lo que ocurre con técnicas basadas en redes bayesianas [119], modelos de mezclas [47, 235], técnicas de generación de reglas [117, 166, 196], técnicas basadas en patrones muestrales [105], etc.

  Comentábamos anteriormente que las redes bayesianas y los modelos de mezcla aproximan la función de densidad conjunta de todos los atributos del problema que tratamos de modelar. A partir de esta densidad conjunta, y mediante marginalización, podemos obtener las densidades condicionales de cualquier subconjunto de atributos dado cualquier otro subconjunto de atributos de valor conocido. En las técnicas de obtención de reglas, de forma general, se pueden generar reglas cuyos antecedentes y consecuentes estén formados por subconjuntos de atributos distintos y, por tanto, obtener información acerca de distintos atributos según sea el conjunto de reglas disparadas o seleccionadas ante la información de entrada. En las técnicas basadas en patrones muestrales está claro que al carecer de fase de aprendizaje, disponemos de toda la información posible a la hora de realizar una nueva inferencia. Dado que estas técnicas están basadas en vecindad, se puede realizar una estimación de cualquier atributo desconocido en un vector de entrada.

- Mediante una dependencia parcial, normalmente de un atributo con respecto a los demás.

  Así, para cada atributo a inferir, es necesario aprender un modelo. La mayoría de las técnicas obtienen modelos que capturan dependencias parciales. Esto ocurre, por ejemplo, con las técnicas de construcción de árboles [224, 225], redes neuronales [118], técnicas basadas en funciones discriminantes [105, 118, 165], etc.

  En el caso de los árboles de clasificación/regresión, hay un atributo objetivo según el cual se realiza la partición del espacio de características. El cambio de dicho objetivo cambia la partición y, por tanto, el árbol generado. Es por ello que es necesario realizar un aprendizaje para cada posible atributo objetivo en el que estemos interesados. Los otros dos grupos de técnicas, redes neuronales y las basadas en funciones discriminantes, dado que son usadas únicamente en clasificación, su atributo objetivo sólo puede ser uno simbólico o nominal. Tanto la red, como las funciones discriminantes son construidas para proporcionar un posible valor de salida del dominio de dicho atributo. De hecho, ya comentábamos en la subsección anterior que estas técnicas están limitadas a trabajar con atributos numéricos y sólo permite un atributo nominal considerado como clase.

Bibliografía

[1] E. Aarts and J. Korst, *Simulated annealing and boltzmann machines: a stochastic approach to combinatorial optimization and neural computing*, John Wiley & Sons, Inc., New York, NY, USA, 1989.

[2] G. Acampora, J.M. Cadenas, V. Loia, and E. Muñoz, *Achieving memetic adaptability through machine learning*, IEEE Transactions on Evolutionary Computation (2010), Sometido.

[3] ———, *A multi-agent memetic system for human-based knowledge selection*, IEEE Transaction on Systems, Man and Cybernetics, part A: Systems and Humans (2010), sometido.

[4] G. Acampora, G. Fenza, E. Muñoz, and B. Romera, *Mejoras en el uso de aprendizaje activo con árboles fuzzy: un ejemplo de su aplicación en la toma de decisiones de un sistema coordinado*, Actas congreso ESTYLF 2010, 2010, pp. 643–648.

[5] G. Acampora, M. Gaeta, V. Loia, P. Ritrovato, and S. Salerno, *Optimizing learning path selection through memetic algorithms*, IJCNN, 2008, pp. 3869–3875.

[6] G. Acampora, V. Loia, J.M. Cadenas, E. Muñoz, R. De Prisco, and R. Zaccagnino, *Un compositor musical automático basado en una metaheurística cooperativa*, III SIMPOSIO SOBRE LÓGICA FUZZY Y SOFT COMPUTING, LFSC'2010, 2010, p. pendiente de publicación.

[7] E. Alba (ed.), *Parallel metaheuristics, a new class of algorithms*, John Wiley and Sons, 2005.

[8] E. Alba, G. Luque, J. García, G. Ordoñez, and G. Leguizamon, *Mallba a software library to design efficient optimisation algorithms*, Int. J. Innov. Comput. Appl. **1** (2007), no. 1, 74–85.

[9] E. Alba, G. Luque, and F. Luna, *Parallel metaheuristics for workforce planning*, Journal of Mathematical Modelling and Algorithms **6** (2007), no. 3, 509–528.

[10] G. Albano, M. Gaeta, and S. Salerno, *E-learning: a model and process proposal*, Int. J. Knowledge Learning **2** (2006), no. 1/2, 73–88.

[11] F.C. Anderson and M.G. Pandy, *Dynamic optimization of human walking*, Journal of Biomechanical Engineering **123** (2001), no. 5, 381–390.

[12] D. Angus and T. Hendtlass, *Dynamic ant colony optimisation*, Applied Intelligence **23** (2005), no. 1, 33–38.

[13] B. Arfi, *Linguistic fuzzy-logic game theory*, J. of Conflict Resolution **50** (2006), no. 1, 28–57.

[14] G. Ausiello, P. Crescenzi, Gambosi, Marchetti A. Spaccamela, and M. Protasi, *Complexity and approximation: Combinatorial optimization problems and their approximability properties*, Springer-Verlag, 1999.

[15] R. Azencott, *Simulated annealing: Parallelization techniques*, Wiley, 1992.

[16] T. Bäck, D. Fogel, and Z. Michalewicz, *Handbook of evolutionary computation*, Institute of Physics Publishing and Oxford University Press, 1997.

[17] R. Baños, C. Gil, J. Ortega, and F. G. Montoya, *A parallel multilevel metaheuristic for graph partitioning*, Journal of Heuristics **10** (2004), no. 3, 315–336.

[18] R. Batta and U.S. Palekar, *Mixed planar/network facility location problems*, Computational Operations Research (1988), 61–67.

[19] R. Battiti, M. Brunato, and P. Campigotto, *Learning while optimizing an unknown fitness surface*, Learning and Intelligent Optimization: Second International Conference, LION 2007 II, Trento, Italy, December 8-12, 2007. Selected Papers (Berlin, Heidelberg), Springer-Verlag, 2008, pp. 25–40.

[20] R. Battiti, M. Brunato, and F. Mascia, *Reactive search and intelligent optimization*, Operations research/Computer Science Interfaces, vol. 45, Springer Verlag, 2008.

[21] B. Bauer, G.P. Müller, and J. Odell, *Agent UML: A Formalism for Specifying Multiagent Software Systems*, International Journal of Software Engineering and Knowledge Engineering **11** (2001), no. 3, 207–230.

[22] J.E. Beasley, *Obtaining test problems via internet*, Journal of Global Optimization **8** (1996), no. 4, 429–433.

[23] R. Beier and B. Voecking, *An experimental study of random knapsack problems*, Algorithmica **45** (2006), no. 1, 121–136.

[24] D. Ben-Arieh and C. Zhifeng, *Linguistic labels aggregation and consensus measure for autocratic decision-making using group recommendations*, IEEE Transactions on

Systems, Man, and Cybernetics Part A - Systems and Humans **36** (2006), no. 3, 558–568.

[25] J.M. Benitez, J.C. Martin, and C. Roman, *Using fuzzy number for measuring quality of service in the hotel industry*, Tourism Management **28** (2007), no. 2, 544–555.

[26] M. Bereau and B. Dubuisson, *A fuzzy extended k-nearest neightbors rule*, Fuzzy Sets and Systems **44** (1991), 17–32.

[27] J.A. Biles, *Genjam: evolution of a jazz improviser*, Creative evolutionary systems, 2002, pp. 165–187.

[28] T. Blackwell, *Particle swarm optimization in dynamic environments*, Evolutionary Computation in Dynamic and Uncertain Environments, Studies in Computational Intelligence, Springer, 2007, pp. 29–49.

[29] T. Blackwell and J. Branke, *Multiswarms, exclusion, and anti-convergence in dynamic environments*, IEEE Transactions on Evolutionary Computation **10** (2006), no. 4, 459–472.

[30] A. Blanco, D.A. Pelta, and J.L. Verdegay, *A fuzzy valuation-based local search framework for combinatorial problems*, Fuzzy Optimization and Decision Making **1** (2002), no. 2, 177–193.

[31] C. Blum, *Ant colony optimization: Introduction and recent trends*, Physics of Life Reviews **2** (2005), no. 4, 353 – 373.

[32] C. Blum, A. Roli, and E. Alba, *An introduction to metaheuristic techniques*, Parallel Metaheuristics, A New Class of Algorithms, John Wiley and Sons, 2005, pp. 3–42.

[33] R.P. Bonasso, R.J. Firby, E. Gat, D. Kortenkamp, D.P. Miller, and M.G. Slack, *Experiences with an architecture for intelligent, reactive agents*, Journal of Experiences and Theory in Artificial Intelligence **9** (1997), no. 2-3, 237–256.

[34] P.P Bonissone and K.S. Decker, *Selecting uncertainty calculi and granularity: An experiment in trading-off precision and complexity*, Uncertainty in Artificial Intelligence, L.H. Kanal and J.F. Lemmer, Eds. (North–Holland), 1986, pp. 217–247.

[35] J. Bradshaw (ed.), *Software Agents*, MIT Press, 1997.

[36] M.L. Brandeau and S.S. Chiu, *An overview of representative problems in location research*, Management Science **35** (1989), 645–674.

[37] J. Branke, *Memory enhanced evolutionary algorithms for changing optimization problems*, Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on, vol. 3, 1999, p. 1882 Vol. 3.

[38] ———, *Evolutionary optimization in dynamic environments*, Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[39] J. Branke, T. Kauler, C. Schmidt, and H. Schmeck, *A multi-population approach to dynamic optimization problems*, Adaptive Computing in Design and Manufacturing: Selected Papers from ACDM'00, Springer, 2000, pp. 100–115.

[40] L. Breiman, J. Friedman, C. Stone, and E. Olshen, *Classification and regression trees*, Wadsworth International Group, 1984.

[41] W. Brenner, H. Wittig, and R. Zarnekow, *Intelligent software agents: Foundations and applications*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.

[42] C. Burgues, *A tutorial on support vector machines for pattern recognition*, Data mining and Knowledge Discovery **2(2)** (1998), 121–167.

[43] E.K. Burke, B.L. Maccarthy, S. Petrovic, and R. Qu, *Knowledge discovery in hyper-heuristic using case-based reasoning on course timetabling*, Selected Papers from the 4 th International Conference on the Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science 2740, 2003, pp. 90–103.

[44] R. Caballero, M. González, F.M. Guerrero, J. Molina, and C. Paralera, *Solving a multiobjective location routing problem with a metaheuristic based on tabu search. application to a real case in andalusia*, European Journal of Operational Research **177** (2007), no. 3, 1751 – 1763.

[45] J.M. Cadenas, J.V. Carrillo, M.C. Garrido, and E. Muñoz, *Nip1.5: Una herramienta software para la generación de conjuntos de datos con imperfección para minería de datos*, Actas del congreso ESTYLF 2010, 2010, pp. 411–416.

[46] J.M. Cadenas, J.V. Carrillo, M.C. Garrido, E. Muñoz, and R. Martinez, *A tool to handle imperfect information in datasets*, University of Murcia, Faculty of Computer Science, 2008.

[47] J.M. Cadenas and M.C. Garrido, *Imperfección explícita versus adaptación de la información en mfgn extendido*, XI Congreso Español sobre Tecnologías y Lógica Fuzzy, 2002, pp. 547–552.

[48] J.M. Cadenas, M.C. Garrido, and J.J. Hernandez, *Improving fuzzy pattern matching techniques to deal with non discrimination ability features*, IEEE International Conference on Systems, Man & Cibernetics, 2004, pp. 5708–5713.

[49] J.M. Cadenas, M.C. Garrido, Hernández L.D., and Muñoz E., *Towards the definitions of a data mining process based in fuzzy sets for cooperative methaeuristics systems*,

Proceedings of the 11th information processing and management of uncertainty in knowlegde-based systems international conference (IPMU 2006) (París (France)), 2006, pp. 2828–2834.

[50] J.M. Cadenas, M.C. Garrido, R. Martínez, and E. Muñoz, *An algorithm to generate optimized fuzzy partitions to classification*, Proceedings of the International Conference on Fuzzy Computation, 2010, pp. 1–10.

[51] J.M. Cadenas, M.C. Garrido, and R. Martínez-España, *Una estrategia de particionamiento fuzzy basada en combinación de algoritmos*, Proceedings de la XIII Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA 2009) (Sevilla (España)), 2009, pp. 379–388.

[52] J.M. Cadenas, M.C. Garrido, and E. Muñoz, *A cooperative system of metaheuristics*, Proceedings of the 7th International Conference on Hybrid Intelligent Systems, 2007, pp. 1–6.

[53] ———, *Construction of a cooperative metaheuristics systems based on data mining and soft-computing: Methodological issues*, Proceeding of the IPMU 2008, 2008, pp. 1246–1253.

[54] ———, *Impact of fuzzy logic in the cooperation of metaheuristics*, New Challenges in Applied Intelligence Technologies, 2008, pp. 225–234.

[55] ———, *Using machine learning in a cooperative hybrid parallel strategy of metaheuristics*, Information Sciences **179** (2009), no. 19, 3255 – 3267.

[56] ———, *Facing dynamic optimization using a cooperative metaheuristic configured via fuzzy logic and svms*, Applied Soft Computing (2010), sometido.

[57] ———, *Una estrategia cooperativa adaptativa en entornos dinámicos*, VII Congreso Español sobre MetaheurÃíticas, Algoritmos Evolutivos y Bioinspirados, MAEB2010, 2010, p. pendiente de publicación.

[58] J.M. Cadenas, M.C. Garrido, E. Muñoz, C. Cruz, D. Pelta, and J.L. Verdegay, *Different approaches for cooperation with metaheuristics*, Encyclopedio of Artificial Intelligence, 2008, pp. 480–487.

[59] J.M. Cadenas, M.C. Garrido, E. Muñoz, and B. Romera, *Un sistema híbrido de metaheurísticas en entornos dinámicos*, VI Congreso Español sobre Metaheurísticas, Algoritmos Genéticos y Algoritmos Evolutivos (MAEB 2009), 2009, pp. 365–372.

[60] J.M. Cadenas, M.C. Garrido, E. Muñoz, and E. Serrano, *Hacia una plataforma de ayuda a la mineria de datos en entornos imperfectos*, V Taller de Minería de Datos y Aprendizaje, TAMIDA'2007, 2007, pp. 247–256.

[61] ———, *Minería de datos aplicada a la modelización de un sistema híbrido*, V Taller de Minería de Datos y Aprendizaje, TAMIDA'2007, 2007, pp. 177–194.

[62] ———, *Un prototipo del coordinador de un sistema metaheurístico cooperativo para el problema de la mochila*, V Congreso Español sobre Metaheurísticas, Algoritmos Genéticos y Algoritmos Evolutivos (MAEB 2007), 2007, pp. 811–818.

[63] J.M. Cadenas, M.C. Garrido, and Mu noz E., *A hybrid system of nature inspired metaheuristics*, Nature Inspired Cooperative Strategies for Optimization, 2008, pp. 95–104.

[64] J.M. Cadenas, A.D. Masegosa, E. Muñoz, and D. Pelta, *Using knowledge discovery in cooperative strategies: two case studies*, Nature Inspired Cooperative Strategies for Optimization (NICSO 2010), Studies in Computational Intelligence, Springer Berlin, 2010, p. In press.

[65] S. Cahon, N. Melab, and E.G. Talbi, *Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics*, Journal of Heuristics **10** (2004), no. 3, 357–380.

[66] J. Campbell, A. Ernst, and Krishnamoorthy M., *Hub location problems*, Facility Location: Applications and Theory, Springer, 2002, pp. 373–406.

[67] L. Canós and V. Liern, *La agregación de la información para la toma de decisiones en la empresa*, XIV Jornadas de la ASEPUMA (Badajoz (Spain)), 2006.

[68] E. Cantú-Paz, *A survey of parallel genetic algorithms*, Calculateurs Paralelles **10** (1998), 1–30.

[69] H. Cardot, *Testing hypotheses in the functional linear model*, Scandinavian Journal of Statistics **30** (2003), no. 1, 241–255.

[70] W. Cedeno and V.R. Vemuri, *On the use of niching for dynamic landscapes*, IEEE International Conference on Evolutionary Computation, 13-16 1997, pp. 361 –366.

[71] J.W. Chinneck (editor-in Chief), *Informs journal of computing*, INFORMS Society, 1987, `http://www.informs.org/Journal/IJOC/`.

[72] P. Clark and R. Boswell, *Rule induction with cn2: some recent improvements*, Fifth European Working Session on Learning, 1991, pp. 151–163.

[73] H.G. Cobb and J.J. Grefenstette, *Genetic algorithms for tracking changing environments*, Proceedings of the 5th International Conference on Genetic Algorithms (San Francisco, CA, USA), Morgan Kaufmann Publishers Inc., 1993, pp. 523–530.

[74] L. Cooper, *Location-allocation problems*, Operations Research **11** (1963), 331–343.

[75] D. Cope, *Experiments in musical intelligence*, A-R Editions, 1996.

[76] ———, *The algorithmic composer*, A-R Editions, 2000.

[77] ———, *Virtual music*, MIT Press, 2004.

[78] D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, and K.V. Price (eds.), *New ideas in optimization*, McGraw-Hill Ltd., UK, Maidenhead, UK, England, 1999.

[79] G. Cornuejols, G. L. Nemhauser, and L.A. Wolsey, *The uncapacited facility location problem*, Discrete Location Theory, John Wiley and Sons Inc., 1990, pp. 119–171.

[80] C. Cortes and V. Vapnik, *Support vector networks*, Machine Learning **20** (1995), no. 3, 273–297.

[81] C. Cotta, *A study of hybridisation techniques and their application to the design of evolutionary algorithms*, AI Commun. **11** (1998), no. 3,4, 223–224.

[82] C. Cotta and P. Moscato, *Una introducción a los algoritmos meméticos*, Revista Iberoamericana de Inteligencia Artificial **19** (2003), 131–148.

[83] C. Cotta, E-g. Talbi, and E. Alba, *E.: Parallel hybrid metaheuristics*, Parallel Metaheuristics, a New Class of Algorithms, John Wiley, 2005, pp. 347–370.

[84] T. G. Crainic and M. Toulouse, *Parallel metaheuristics*, Fleet Management and Logistics (T. G. Crainic and G. Laporte, eds.), Kluwer Academic Publisher, 1998, pp. 205–251.

[85] T.G. Crainic, *Parallel solution methods for vehicle routing problems*, The Vehicle Routing Problem: Latest Advances and New Challenges, Springer, 2008, pp. 171–198.

[86] T.G. Crainic, B. Di Chiara, M. Nonato, and L. Tarricone, *Tackling electrosmog in completely configured 3g networks by parallel cooperative meta-heuristics*, IEEE Wireless Communications **13** (2006), no. 6, 34 –41.

[87] T.G. Crainic and M. Gendreau, *Cooperative parallel tabu search for capacitated network design*, Journal of Heuristics **8** (2002), no. 6, 601–627.

[88] T.G. Crainic, M. Gendreau, P. Hansen, and N. Mladenović, *Cooperative parallel variable neighborhood search for the p-median*, Journal of Heuristics **10** (2004), no. 3, 293–314.

[89] T.G. Crainic and M. Toulouse, *Parallel strategies for metaheuristics*, Handbook of Metaheuristics (International Series in Operations Research & Management Science), F.W. Glover, and G.A. Kochenberger, Eds (Springer), 2003.

[90] T.G. Crainic, M. Toulouse, and M. Gendreau, *Toward a taxonomy of parallel tabu search heuristics*, INFORMS Journal on Computing **9** (1997), no. 1, 61–72.

[91] V-D. Cung, S.L. Martins, C.C. Ribeiro, and C. Roucairol, *Strategies for the parallel implementation of metaheuristics*, Essays and Surveys in Metaheuristics, Kluwer Academic Publishers, 2002, pp. 263–308.

[92] D. Dagger, A. O'Connor, S. Lawless, E. Walsh, and V.P. Wade, *Service-oriented e-learning platforms: From monolithic systems to flexible services*, IEEE Internet Computing **11** (2007), no. 3, 28–35.

[93] R. De Prisco and R. Zaccagnino, *An evolutionary music composer algorithm for bass harmonization*, EvoWorkshops, 2009, pp. 567–572.

[94] K. Decker, A. Pannu, K. Sycara, and M. Williamson, *Designing behaviors for information agents*, AGENTS '97: Proceedings of the first international conference on Autonomous agents, ACM, 1997, pp. 404–412.

[95] A. Dempster, N. Laird, and D. Rubin, *Maximum likelihood estimation from incomplete data via the em algorithm*, Journal of the Royal Statistical Society **39(1)** (1977), 1–38.

[96] G. Di Caro, L.M. Gambardella, and M. Dorigo, *Ant algorithms for discrete optimization*, Artificial Life **5** (1999), 137–172.

[97] B.A Díaz and K.A. Dowsland, *Diseño de heurísticas y fundamentos del recocido simulado*, Revista Iberoamericana de Inteligencia Artificial **19** (2003), 93–102.

[98] D. Dicheva and C. Dichev, *Tm4l: Creating and browsing educational topic maps*, British Journal of Educational Technology **37** (2006), no. 3, 391–404.

[99] R. Dodhiawala, V. Jagannathan, and L.S. Baum, *Blackboard architectures and applications*, Academic Press, Inc., Orlando, FL, USA, 1989.

[100] M. Dorigo and T. Stützle, *Ant colony optimization metaheuristic*, Handbook of Metaheuristics, Kluwer Academic, 2003.

[101] W. Du and B. Li, *Multi-strategy ensemble particle swarm optimization for dynamic optimization*, Inf. Sci. **178** (2008), no. 15, 3096–3109.

[102] D. Dubois and C. Prade, H.and Testmale, *Weighted fuzzy pattern matching*, Fuzzy Sets and Systems **28** (1988), 313–331.

[103] D. Dubois and H. Prade, *A review of fuzzy set aggregation connectives*, Information Sciences **36** (1985), no. 1-2, 85–121.

[104] ———, *Possibility theory: An approach to computerized processing of uncertainty*, Plenium Press, New York, 1988.

[105] R. Duda, P. Hart, and D. Stork, *Pattern classification*, John Wiley & Sons, 2001.

[106] K. Dwyer and R. Holte, *Decision tree instability and active learning*, ECML '07: Proceedings of the 18th European conference on Machine Learning, 2007, pp. 128–139.

[107] S. Dzeroski, *Multi-relational data mining: An introduction*, ACM 2003, 2003, pp. 1–16.

[108] K. Ebcioglu, *An expert system for harmonizing four-part chorales*, Machine models of music (Cambridge, MA, USA), MIT Press, 1992, pp. 385–401.

[109] R.C. Eberhart, Y. Shi, and J. Kennedy, *Swarm intelligence (the morgan kaufmann series in artificial intelligence)*, 1st ed., Morgan Kaufmann, April 2001.

[110] M. Ebner, *E-learning 2.0 = e-learning 1.0 + web 2.0?*, Availability, Reliability and Security, International Conference on **0** (2007), 1235–1239.

[111] R.W. Eglese, *Simulated annealing: A tool for operational research*, European Journal of Operational Research **46** (1990), no. 3, 271–281.

[112] A.E. Eiben, M. Horvath, W. Kowalczyk, and M.C. Schut, *Reinforcement learning for online control of evolutionary algorithms*, LNAI - Engineering Self-Organising Systems, Springer Berlin, 2007, pp. 151–160.

[113] M. El-Abd and M. Kamel, *A taxonomy of cooperative search algorithms*, Hybrid Metaheuristics, 2005, pp. 32–41.

[114] T. Finin, Y. Labrou, and J. Mayfield, *Kqml as an agent communication language*, Software Agents (Jeffrey Bradshaw, ed.), MIT Press, 1997, pp. 291–316.

[115] FIPA, *FIPA ACL Message Structure Specification*, Tech. report, Foundation for Intelligent Physical Agents, 2002, Available at `http://www.fipa.org/specs/fipa00061`.

[116] ———, *FIPA Interaction Protocol Library Specification*, Tech. report, Foundation for Intelligent Physical Agents, 2003, Available at `http://www.fipa.org/specs/fipa00025`.

[117] D. Fisher, *Knoweledge adquisition via incremental conceptual clustering*, Machine Learning **2** (1987), 139–172.

[118] J.A. Freeman and D.M. Skapura, *Redes neuronales: Algoritmos, aplicaciones y técnicas de programación*, Addison-Wesley/Diaz de Santos, 1993.

[119] N. Friedman, D. Greiger, and Goldszmidt D., *Bayesian network classifiers.*, Machine Learning **29** (2003), 103–130.

[120] M. Gagliolo and J. Schmidhuber, *Gambling in a computationally expensive casino: Algorithm selection as a bandit problem*, NIPS 2006 Workshop on Online Trading of Exploration and Exploitation, 2006.

[121] ———, *Learning dynamic algorithm portfolios*, Annals of Mathematics and Artificial Intelligence **47** (2006), no. 3-4, 295–328.

[122] D. Gang, D. Lehman, and N. Wagner, *Harmonizing melodies in real time: The connectionist approach*, Proc. of the 1997 International Computer Music Conference, 1997, pp. 27–31.

[123] S. García, A. Fernández, J. Luengo, and F. Herrera, *A study statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability*, Soft Computing - A Fusion of Foundations, Methodologies and Applications **13** (2009), no. 10, 959–977.

[124] M. Gendreau, T.G. Crainic, P. Hansen, and N. Mladenovic, *Cooperative parallel variable neighborhood search for the p-median*, Journal of Heuristics **10** (2004), no. 3, 293–314.

[125] R. Gilad-Bachrach, A. Navot, and N. Tishby, *Query by committee made real*, NIPS, 2005.

[126] F. Glover, *Future paths for integer programming and links to artificial intelligence*, Coputers and Operation Researchl **5** (1986), 533–549.

[127] F. Glover and G. Kochenberger, *Handbook of metaheuristics*, Kluwer Academic Publishers, 2003.

[128] F. Glover and M. Laguna, *Tabu search*, Kluwer Academic, 1997.

[129] F. Glover and B. Melián, *Busqueda tabú*, Revista Iberoamericana de Inteligencia Artificial **19** (2003), 29–48.

[130] D.E. Goldberg and R.E. Smith, *Nonstationary function optimization using genetic algorithm with dominance and diploidy*, Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application (Hillsdale, NJ, USA), L. Erlbaum Associates Inc., 1987, pp. 59–68.

[131] J.R. González, A.D. Masegosa, and I.J. García, *A cooperative strategy for solving dynamic optimization problems*, Memetic Computing (2009), 1–12.

[132] L. Gosselin, M. Tye-Gingras, and F. Mathieu-Potvin, *Review of utilization of genetic algorithms in heat transfer problems*, International Journal of Heat and Mass Transfer **52** (2009), no. 9-10, 2169 – 2188.

[133] M. Grabisch, *On equivalence classes of fuzzy connectives – the case of fuzzy integrals*, IEEE transaction Fuzzy Systems **3** (1995), 96–109.

[134] D.R. Greening, *Parallel simulated annealing techniques*, Phys. D **42** (1990), no. 1-3, 293–306.

[135] J. Grefenstette, *Genetic algorithms for changing environments*, Parallel Problem Solving from Nature, 2: Proceedings of the Second Conference on Parallel Problem Solving from Nature, Brussels, Belgium, 28-30 September, 1992 (New York, NY, USA) (Reinhard Manner and Bernard Manderick, eds.), Elsevier Science Inc., 1992.

[136] T.R. Gruber, *Toward principles for the design of ontologies used for knowledge sharing*, Int. J. Hum.-Comput. Stud. **43** (1995), no. 5-6, 907–928.

[137] H. Guo, *A bayesian approach for automatic algorithm selection*, IJCAI03 Workshop on AI and Autonomic Comp., 2003, pp. 1–5.

[138] H. Guo and W.H. Hsu, *A machine learning approach to algorithm selection for np-hard optimization problems: a case study on the mpe problem*, Annals of Operations Research **156** (2007), no. 1, 61–82.

[139] P. Hansen and N. Mladenovic, *Metaheuristics: Advances and trends in local search procedures for optimization*, Kluwer Academic, 1999.

[140] ———, *Variable neighbourhood search*, Handbook of Metaheuristics, Kluwer Academic, 2003.

[141] P. Hansen, N. Mladenovic, and J.A. Moreno, *Búsqueda de entorno variable*, Revista Iberoamericana de Inteligencia Artificial **19** (2003), 77–92.

[142] P. Hansen and Mladenovic N., *Variable neighborhood search: Principles and applications*, European Journal of Operational Research **130** (2001), 449–467.

[143] W.E. Hart, N. Krasnogor, and J.E. Smith (eds.), *Recent advances in memetic algorithms*, Studies in Fuzzyness and Soft Computing, vol. 166, Springer Berlin Heidelberg New York, 2004, ISBN 3-540-22904-3.

[144] F. Hayes-Roth, D.A. Waterman, and D.B. Lenat, *Building expert systems*, Addison-Wesley Longman Publishing Co., Inc., 1983.

[145] D. Henderson, S.H. Jacobson, and A.W. Johnson, *The theory and practice of simulated annealing*, Handbook of Metaheuristics, Kluwer Academic, 2003.

[146] J. Hernández, M.J. Ramírez, and Ferri C., *Introducción a la minería de datos*, Pearson, Prentice Hall, España, 2004.

[147] F. Herrera, E. Herrera-Viedma, and L. Martínez, *A fusion approach for managing multigranularity linguistic terms sets in decision making*, Fuzzy Sets and Systems **114** (2000), no. 1, 43–58.

[148] F. Herrera, E. Herrera-Viedma, and L. Martinez, *A fuzzy linguistic methodology to deal with unbalanced linguistic term sets*, IEEE Transactions on Fuzzy Systems **16** (2008), no. 2, 354–370.

[149] F. Herrera, E. Herrera-Viedma, and J.L. Verdegay, *Direct approach processes in group decision making using linguistic owa operators*, Fuzzy Sets and Systems **79** (1996), no. 2, 175–190.

[150] F. Herrera and L. Martínez, *A 2-tuple fuzzy linguistic representation model for computing with words*, IEEE Transactions on Fuzzy Systems **8** (1999), no. 6, 746–752.

[151] E. Herrera-Viedma, F. Herrera, L. Martinez, J.C. Herrera, and A.G. Lopez-Herrera, *Incorporating filtering techniques in a fuzzy linguistic multi-agent model for gathering of information on the web*, Fuzzy Sets and Systems **148** (2004), no. 1, 61–83.

[152] E. Herrera-Viedma, L. Martínez, F. Mata, and F. Chiclana, *A consensus support system model for group decision-making problems with multi-granular linguistic preference relations*, IEEE Transaction on Fuzzy Systems **13** (2005), no. 5, 644–658.

[153] A. Hiller, *Computer music*, Scientific American **201** (1959), no. 6, 109–120.

[154] L. Hiller and L.M. Isaacson, *Experimentalmusic*, McGraw-Hill, 1959.

[155] A. Horner and L. Ayers, *Harmonization of musical progressions with genetic algorithms*, proceedings of the 1995 International Computer Music Conference, 1995, pp. 483–484.

[156] J. Huang, S. Ertekin, Y. Song, H. Zha, and C.L. Giles, *Efficient multiclass boosting classification with active learning*, SDM, 2007.

[157] W. Isard, *Location and space economy*, Technology Press, MIT, 1962.

[158] J. Jacob, *Composing with genetic algorithms*, Proc. International Computer Music Conference (ICMC '95) (Orlando (USA)), 1994, pp. 452–455.

[159] C.Z. Janikow, *Fuzzy decision trees: issues and methods*, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics **28** (1998), no. 1, 1 –14.

[160] N.R. Jennings, K.P. Sycara, and M. Wooldridge, *A roadmap of agent research and development*, Autonomous Agents and Multi-Agent Systems **1** (1998), no. 1, 7–38.

[161] L. Jourdan, C. Dhaenens, and E.G. Talbi, *Using datamining techniques to help metaheuristics: A short survey*, Hybrid Metaheuristics, 2006, pp. 57–69.

[162] J.M. Juarez, J.J Cañadas, and R. Marín, *Control*, Inteligencia artificial: técnicas, métodos y aplicaciones, 2005, pp. 589–646.

[163] A. Karaman, S. Uyar, and G. Eryigit, *The memory indexing evolutionary algorithm for dynamic environments*, Applications on Evolutionary Computing, Springer, 2005, pp. 563–573.

[164] J.M. Keller, M.R. Gray, and J.A. Givens, *A fuzzy k-nearest neighbor algorithm*, IEEE Transaction on System, Man, and Cybernetics **15** (1985), no. 4, 580–585.

[165] J.M. Keller and D.J. Hunt, *Incorporating fuzzy membership functions into the perceptron algorithm*, IEEE Transaction on Pattern Analysis and Machine Intelligence **7** (1985), no. 6, 693–699.

[166] T. Kohonen, *Self-organising maps*, Springer-Verlag, 1995.

[167] T.J. Kopcha and H. Sullivan, *Learner preferences and prior knowledge in learner-controlled computer-based instruction*, Educational Technology Research and Development **56** (2008), no. 3, 265–286.

[168] R. Koper, *Editorial: Current research in learning design*, Educational Technology & Society **9** (2006), no. 1, 13–22.

[169] J. Kratica, Z. Stanimirovic, D. Tosic, and V. Filipovic, *Two genetic algorithms for solving the uncapacitated single allocation p-hub median problem*, European Journal of Operational Research **182** (2007), no. 1, 15 – 28.

[170] L.I. Kuncheva, *Fuzzy vs non-fuzzy in combining classifiers designed by boosting*, IEEE Transactions on Fuzzy Systems **11** (2003), no. 6, 729–741.

[171] Y. Labrou and T. Finin, *Semantics and conversations for an agent communication language*, Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97), Morgan Kaufmann, 1997, pp. 235–242.

[172] M. Laguna and P. Moscato, *Optimización heurística y redes neuronales*, Paraninfo S.A., 1996.

[173] M. Laguna (editor-in Chief), *Journal of heuristic*, Springer, 2004, `http://www.editorialmanager.com/heur/`.

[174] G. Laporte and I.H. Osman, *Metaheuristics: A bibliography*, Annals of Operations Research **63** (1996), 63–623.

[175] A. Le Bouthillier and T.G. Crainic, *A cooperative parallel meta-heuristic for the vehicle routing problem with time windows*, Computers & Operations Research **32** (2005), no. 7, 1685 – 1708.

[176] K. Lee and S. Lee, *Efficient parallelization of simulated annealing using multiple markov chains: an application to graph partitioning*, Proceedings of the International Conference on Parallel Processing, 1992, pp. 177–180.

[177] H. Liu, H. Hussain, C.L. Tan, and M. Dash, *Discretization: an enabling technique*, Journal of Data Mining and Knowledge Discovery **6** (2002), no. 4, 293–423.

[178] H. Liu and R. Setiono, *Incremental feature selection*, Applied Intelligence **9** (1998), 217–230.

[179] Y. Liu, *Active learning with support vector machine applied to gene expression data for cancer classification*, J. Chemistry Information and Computer Science **44** (2004), 1936–1941.

[180] R.F. Love, J.G. Morris, and G.O. Wesolowsky, *Facilities location: Models & methods. publications in operations research series*, vol. 7, North-Holland, 1988.

[181] G. Loy, *Musimathics: The mathematical foundations of music, volume 1*, The MIT Press, 2006.

[182] M. Luck, P. McBurney, and C. Preist, *Agent technology: Enabling next generation computing: a roadmap for agent based computing*, Agentlink, 2003.

[183] J. Ma, D. Ruan, Y. Xu, and G. Zhang, *A fuzzy-set approach to treat determinacy and consistency of linguistic terms in multi-criteria decision making*, International Journal of Approximate Reasoning **44** (2007), no. 2, 165–181.

[184] S. Martello, D. Pisinger, and P. Toth, *Dynamic programming and strong bounds for the 0-1 knapsack problem*, Manage. Sci. **45** (1999), no. 3, 414–424.

[185] O.C. Martin, H.R. LourenÃ§o, and T. Stützle, *Iterated local search*, Handbook of Metaheuristics, Kluwer Academic, 2003, pp. 321–354.

[186] R.A. McIntyre, *Bach in a box: The evolution of four part baroque harmony using the genetic algorithm*, International Conference on Evolutionary Computation, 1994, pp. 852–857.

[187] G.J. McLachlan and T. Krishnan, *The em algorithm and extensions*, Wiley Series in Probability and Statistics, 1997.

[188] B. Melián, J.A. Moreno, and J.M. Moreno, *Metaheurísticas: un visión global*, Revista Iberoamericana de Inteligencia Artificial **19** (2003), 7–28.

[189] A.A.M. Meneses, M.D. Machado, and R. Schirru, *Particle swarm optimization applied to the nuclear reload problem of a pressurized water reactor*, Progress in Nuclear Energy **51** (2009), no. 2, 319 – 326.

[190] N.H. Michael and P.S. Munindar, *Cognitive Agents*, IEEE Internet Computing **2** (1998), no. 6, 87–89.

[191] R.S. Michalski, I. Mozetic, J. Hong, and N. Lavrac, *The aq15 inductive learning system: an overview and experiments*, Proc. 1st Intl. Meeting on Advances in Learning, 1986, pp. 1–33.

[192] M. Milano and A. Roli, *Magma: a multiagent architecture for metaheuristics*, IEEE Transactions on Systems, Man, and Cybernetics, Part B **34** (2004), no. 2, 925–941.

[193] E.R. Miranda, *Composing music with computers*, Focal Press, 2001.

[194] S. Mitra and S.K. Pal, *Fuzzy multi-layer perceptron, inferencing and rule generation*, IEEE Trans. on Neural Networks **6** (1995), no. 1, 51–63.

[195] N. Mladenovic, D. Urosevic, D. Pérez-Brito, and C.G. García-González, *Variable neighbourhood search for bandwidth reduction*, European Journal of Operational Research **200** (2010), no. 1, 14 – 27.

[196] J. Moody and C. Darken, *Fast learning in networks of locally tuned processing units*, Neural computation **1** (1989), 281–294.

[197] N. Mori, H. Kita, and Y. Nishikawa, *Adaptation to a changing environment by means of the feedback thermodynamical genetic algorithm*, PPSN V: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature (London, UK), Springer-Verlag, 1998, pp. 149–158.

[198] R.W. Morrison and K.A. De Jong, *A test problem generator for non-stationary environments*, Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on, vol. 3, 1999, p. 2053 Vol. 3.

[199] S.D. Muller, N.N. Schraudolph, and P.D. Koumoutsakos, *Step size adaptation in evolution strategies using reinforcement learning*, IEEE International Conference on E-Commerce Technology, vol. 1, 2002, pp. 151–156.

[200] S. Murugesan, *Understanding web 2.0*, IT Professional **9** (2007), no. 4, 34–41.

[201] D. Nauck and R. Kruse, *A neuro-fuzzy method to learn fuzzy classification rules from data*, Fuzzy Sets and Systems **89** (1996), 277–288.

[202] H.P. Nii, *Blackboard systems: the blackboard model of problem solving and the evolution of blackboard architectures*, Computation & intelligence: collected readings, American Association for Artificial Intelligence, 1995, pp. 447–474.

[203] P. Nii, *Blackboard systems part two: Blackboard application systems*, AI Mag. **7** (1986), no. 3, 82–106.

[204] H.S. Nwana, *Software Agents: An Overview*, Knowledge Engineering Review **11** (1996), no. 3, 205–244.

[205] I.O. Oduntan, M. Toulouse, R. Baumgartner, C. Bowman, R. Somorjai, and T.G. Crainic, *A multilevel tabu search algorithm for the feature selection problem in biomedical data*, Comput. Math. Appl. **55** (2008), no. 5, 1019–1033.

[206] M.E. O'kelly, *A quadratic integer program for the location of interacting hub facilities*, European Journal of Operational Research **32** (1987), no. 3, 393–404.

[207] A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf (eds.), *Coordination of internet agents: Models, technologies, and applications*, Springer, 2001.

[208] G. Osman and G. Laporte, *Metaheuristics: a bibliography*, Annals of Operations Research (1996), no. 63, 513–623.

[209] G. Pampara, A.P. Engelbrecht, and T. Cloete, *Cilib: A collaborative framework for computational intelligence algorithms - part i*, IJCNN, 2008, pp. 1750–1757.

[210] _____, *Cilib: A collaborative framework for computational intelligence algorithms - part ii*, IJCNN, 2008, pp. 1764–1773.

[211] P.M. Pardalos and M.G.C. Resende (eds.), *Handbook of applied optimization*, pub-OXFORD, 2002.

[212] J. Pearl, *Heuristics: intelligent search strategies for computer problem solving*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[213] D. Pelta, C. Cruz, and J.R. González, *A study on diversity and cooperation in a multiagent strategy for dynamic optimization problems*, Int. J. Intell. Syst. **24** (2009), no. 7, 844–861.

[214] D. Pelta, A. Sancho-Royo, C. Cruz, and J.L. Verdegay, *Using memory and fuzzy rules in a co-operative multi-thread strategy for optimization*, Information Sciences **176** (2006), no. 13, 1849 – 1868.

[215] M. Petrik and S. Zilberstein, *Learning parallel portfolios of algorithms*, Annals of Mathematics and Artificial Intelligence **48** (2006), no. 1-2, 85–106.

[216] D. Pisinger, *Where are the hard knapsack problems?*, Computers & Operations Research **32** (2005), 2271–2282.

[217] W. Piston, *Harmony*, W.W. Norton, 1987.

[218] G. Polya, *How to solve it*, Princeton. N.J. Princeton University Press, 1945.

[219] S.T. Powers and J. He, *A hybrid artificial immune system and self organising map for network intrusion detection*, Information Sciences **178** (2008), no. 15, 3024–3042.

[220] A. Predoehl, *The theory of location in its relation to general economics*, Journal of Political Economy **36** (1928), 371–190.

[221] S. Prestwich, *Tuning local search by average-reward reinforcement learning*, Learning and Intelligent Optimization: Second International Conference, LION 2007 II, Trento, Italy, December 8-12, 2007. Selected Papers, 2008, pp. 192–205.

[222] J. Puchinger and G.R. Raidl, *Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification*, Lecture notes in computer science, Springer, 2005, pp. 41–53.

[223] J.R. Quinlan, *Induction on decision trees.*, Machine Learning **1** (1986), 81–106.

[224] _____, *Learning logical definitions from relations*, Machine Learning **5** (1990), 239–266.

[225] _____, *C4.5: programs for machine learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[226] G.R. Raidl, *A unified view on hybrid metaheuristics*, Hybrid Metaheuristics, 2006, pp. 1–12.

[227] D.J. Ram, T.H. Sreenivas, and K.G. Subramaniam, *Parallel simulated annealing algorithms*, J. Parallel Distrib. Comput. **37** (1996), no. 2, 207–212.

[228] A.S. Rao and M.P. Georgeff, *An abstract architecture for rational agents*, Proceedings of the Third International Conference on Principles and Knowledge Representation and Reasoning (KR-92) (Bernhard Nebel, Charles Rich, and William R. Swartout, eds.), Morgan Kaufmann, 1992, pp. 439–449.

[229] C. Reeves, *Guided local search*, Kluwer Academic, 2003.

[230] C.R. Reeves (ed.), *Modern heuristic techniques for combinatorial problems*, John Wiley & Sons, Inc., New York, NY, USA, 1993.

[231] C.C. Ribeiro and O. Hansen (editors), *Essays and surveys in metaheuristic*, Kluwer Academic Publishers, 2002.

[232] J.R. Rice, *The algorithm selection problem*, Advances in Computers **15** (1976), 65–118.

[233] P. Rohlfshagen and X. Yao, *The dynamic knapsack problem revisited: A new benchmark problem for dynamic combinatorial optimisation*, EvoWorkshops 2009: LNCS 5484, 2009, pp. 745–754.

[234] A. Romiszowski, *How's the e-learning baby? factors leading to success or failure of an educational technology innovation*, Educational Technology **44** (2004), no. 1, 5–27.

[235] A. Ruiz, P.E. López de Teruel, and M.C. Garrido, *Probabilistic inference from arbitrary uncertainty using mixtures of factorized generalized gaussians*, Journal of Artificial Intelligent Research **9** (1998), 167–217.

[236] M. Saar-Tsechansky and F. Provost, *Active sampling for class probability estimation and ranking*, Mach. Learn. **54** (2004), no. 2, 153–178.

[237] B. Scholkopf and A. Smola, *Learning with kernel. support vector machines, regularization, optimization, and beyond*, MIT Press, 2002.

[238] R. Schoonderwoerd, J.L. Bruten, O.E. Holland, and L.J.M. Rothkrantz, *Ant-based load balancing in telecommunications networks*, Adapt. Behav. **5** (1996), no. 2, 169–207.

[239] J.R. Searle, *Speech acts : an essay in the philosophy of language*, Cambridge University Press, London, 1969.

[240] R. Swaminathan, Mi.E. Pfund, J.W. Fowler, S.J. Mason, and A. Keha, *Impact of permutation enforcement when minimizing total weighted tardiness in dynamic flowshops with uncertain processing times*, Computers & Operations Research **34** (2007), no. 10, 3055 – 3068.

[241] E.-G. Talbi, *A taxonomy of hybrid metaheuristics*, Journal of Heuristics **8** (2002), no. 5, 541–564.

[242] S. Talukdar, L. Baerentzen, A. Gove, and P.S. de Souza, *Asynchronous teams: Cooperation schemes for autonomous agents*, J. Heuristics **4** (1998), no. 4, 295–321.

[243] S. Talukdar, S. Murti, and R. Akkirayu, *Asynchronous teams*, Handbook of Metaheuristics (International Series in Operations Research & Management Science), F.W. Glover, and G.A. Kochenberger, Eds (Springer), 2003.

[244] F. Thabtah and P. Cowling, *Mining the data from a hyperheuristic approach using associative classification*, Expert Syst. Appl. **34** (2008), no. 2, 1093–1101.

[245] S. Tong and D. Koller, *Support vector machine active learning with applications to text classification*, J. Mach. Learn. Res. **2** (2002), 45–66.

[246] K. Trojanowski and S.T. Wierzchon, *Immune-based algorithms for dynamic optimization*, Inf. Sci. **179** (2009), no. 10, 1495–1515.

[247] E.P.K. Tsang and C. Voudoris, *Guided local search*, Handbook of Metaheuristics, Kluwer Academic, 2003.

[248] R.K. Ursem, B. Filipic, and T. Krink, *Exploring the performance of an evolutionary algorithm for greenhouse control*, Information Technology Interfaces, 2002. ITI 2002. Proceedings of the 24th International Conference on, 2002, pp. 429 – 434 vol.1.

[249] V. Vapnik, *Statistical learning theory*, Wiley, 1998.

[250] J.L. Verdegay, R.R. Yager, and P.P. Bonissone, *On heuristics as a fundamental constituent of soft computing*, Fuzzy Sets and Systems **159** (2008), no. 7, 846–855.

[251] M.G.A. Verhoeven and E.H.L. Aarts, *Parallel local search*, Journal of Heuristics **1** (1995), no. 1, 43,65.

[252] S. Voss, *Tabu search: Applications and prospects*, Network Optimization Problems, D.Z. Du, and P.M. Pardalos, Eds (World Scientific), 1993.

[253] S. Voss, I.H. Osman Martello, and C. Roucariol, *Meta-heuristics. advances and trends in local search paradigms for optimization*, Kluwer Academic Publishers, 1999.

[254] P. Walley, *Statistical reasoning with imprecise probabilities*, Chapman and Hall, London, 1991.

[255] A. Weber, *Theory of the location of industries*, Russell and Russell, 1909.

[256] G. Weiss (ed.), *Multiagent systems: a modern approach to distributed artificial intelligence*, MIT Press, 1999.

[257] G. Weiss (ed.), *Multiagent systems: a modern approach to distributed artificial intelligence*, MIT Press, Cambridge, MA, USA, 1999.

[258] D.J. White, *Heuristic Programming*, IMA J Management Math **2** (1989), no. 2, 173–188.

[259] G. Wiggins, G. Papadopoulos, S. Phon-amnuaisuk, and A. Tuson, *Evolutionary methods for musical composition*, International Journal of Computing Anticipatory Systems **4** (1999), 1–14.

[260] I.H. Witten and E. Frank, *Data mining**, Morgan Kaufmann Publishers, 2000.

[261] M.J. Wooldridge, *Multi-agent Systems : An Introduction*, John Wiley and Sons, 2002.

[262] R.R. Yager, *On ordered weighted averaging aggregation operators in multicriteria decision making*, IEEE transactions on Systems, Man and Cybernetics **18** (1988), no. 1, 183–190.

[263] J. Yang and Y Zhuang, *An improved ant colony optimization algorithm for solving a complex combinatorial optimization problem*, Applied Soft Computing **10** (2010), no. 2, 653 – 660.

[264] H. Yu, *Svm selective sampling for ranking with application to data retrieval*, KDD '05: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining (New York, NY, USA), ACM, 2005, pp. 354–363.

[265] V.F. Yu, S. Lin, W. Lee, and C. Ting, *A simulated annealing heuristic for the capacitated location routing problem*, Computers & Industrial Engineering **58** (2010), no. 2, 288 – 299.

[266] L.A. Zadeh, *Fuzzy sets*, Information and Control **8** (1965), no. 3, 338–353.

[267] ———, *The concept of a linguistic variable and its application to approximate reasoning - i*, Inf. Sci. **8** (1975), no. 3, 199–249.

[268] ———, *The concept of a linguistic variable and its application to approximate reasoning - ii*, Inf. Sci. **8** (1975), no. 4, 301–357.

[269] ———, *The concept of a linguistic variable and its application to approximate reasoning - iii*, Inf. Sci. **9** (1975), no. 1, 43–80.

[270] ———, *Soft computing and fuzzy logic*, IEEE Software **11** (1994), no. 6, 48–56.

[271] ———, *Applied soft computing - foreword*, Applied Soft Computing **1** (2001), no. 1, 1–2.

[272] L.A. Zadeh and J. Kacprzyk, *Fuzzy logic for the management of uncertainty*, John Wiley & Sons, Inc., New York, 1992.

[273] Z. Zhang, *Multiobjective optimization immune algorithm in dynamic environments and its application to greenhouse control*, Appl. Soft Comput. **8** (2008), no. 2, 959–971.