Cooperative Methods in Optimisation: Analysis and Results

PhD Dissertation

Antonio David Masegosa Arredondo

Advisors: David Alejandro Pelta
José Luis Verdegay Galdeano

Universidad de Granada
E.T.S. de Ingenierías Informática y de Telecomunicación
Departamento de Ciencias de la Computación e Inteligencia Artificial

Editor: Editorial de la Universidad de Granada
Autor: Antonio David Masegosa Arredondo
D.L.: GR 2968-2010
ISBN: 978-84-693-2560-8

La memoria titulada **"Cooperative methods in optimisation: analysis and results"**, que presenta D. Antonio David Masegosa Arredondo para optar al grado de Doctor en Informática, ha sido realizada en el Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada, bajo la dirección de los doctores D. David Alejandro Pelta y D. José Luis Verdegay Galdeano, del mismo departamento.

Granada, Febrero de 2010

Antonio David Masegosa Arredondo

David Alejandro Pelta

José Luis Verdegay Galdeano

-A mis padres-
(To my parents)

Contents

Agradecimientos

Han sido tres años y medio en los que se han sucedido los buenos y los malos momentos (por suerte muy pocos) y que para mí han marcado un ciclo académico bastante gratificante. Está claro que hasta aquí no habría podido llegar solo y quiero dedicar las siguientes líneas a dar las gracias a todas aquellas personas e instituciones que con su apoyo y ayuda han hecho posible esta tesis.

Mis primeras palabras de agradecimiento van dirigidas, como no, a mis dos tutores, Curro y David, por darme la oportunidad de embarcarme en esta aventura del doctorado y por haberme guiado en ella todo este tiempo con sus enseñanzas y conocimientos. Me siento realmente afortunado de haberles tenido como tutores porque he podido trabajar con unos excelentes profesionales y mejores personas, que aparte de compañeros de trabajo, han sido y son unos buenos amigos.

En esta relación trabajo/amistad no me puedo dejar a los demás miembros del grupo MODO: Maite, Carlos, Juan Ramón, Nacho, Socorro,... a los que les tengo que agradecer el apoyo, los consejos y el trato recibido desde el primer día. Me gustaría también dar las gracias a Alejandro Sancho por dirigir mis primeros pasos en esta etapa y porque parte de esta tesis ha surgido de sus ideas.

Quiero incluir en estos agradecimientos al Ministerio español de Ciencia e Innovación por financiar mi formación doctoral a través del programa de becas FPI. Dicha financiación también ha hecho posible mi estancia en Nottingham con el profesor Natalio Krasnogor, al que aquí quiero dar las gracias por haberme dado la oportunidad de trabajar con él y con otros excelentes investigadores como son Carlos García, Claudio Lima y Jamie Twycross.

En un ámbito más personal, también hay mucha gente a la que le tengo que reconocer el apoyo y la ayuda recibida. No podían faltar Fran, Javi,

Fernando, Nacho, Alex, Eli y demás amigos/as porque las risas, las cañas, las noches de marcha, las copas,... también ayudan a hacer una tesis.

Por supuesto, en todo este tiempo siempre he tenido el apoyo y el apego de mis hermanas Luisa y Loli y de mi hermano Andrés, quienes sin duda alguna merecen un lugar especial en estos agradecimientos. No me puedo olvidar de que en estos años (y en otros muchos) me he servido de la experiencia y los consejos de mi hermano para hacer más llano mi camino. Por todo ello quiero darle las gracias de nuevo.

Por último, me gustaría agradecer muy especialmente a mis padres todo el esfuerzo y sacrificio que han hecho y el apoyo incondicional que me han ofrecido en todo momento. Es mucho lo que les debo y quiero reconocérselo dedicándoles esta tesis.

Esta tesis ha sido financiada por el Junta de Andalucía con el proyecto TIC-02970, por el Ministerio español de Ciencia e Innovación a través de los proyectos TIN2005-08404-C04-01 y TIN2008-01948, y por la beca FPI BES-2006-13524 de esta última institución.

Acknowledgements

In these three years and a half I have had good and bad moments (very few, fortunately) that have marked a gratifying academic period of my life. It is clear that I could not achieve this goal by myself and I would like to dedicate the next lines to thank those people and institutions whose support and help have made possible this thesis.

My first words of appreciation go to my advisors, Curro and David, for giving me the opportunity of embarking on this doctorate and for guiding me with their teachings and knowledge all this time. I feel really fortunate to have them as advisors because I have been able to work with excellent professionals and better people that apart from workmates have been and are good friends.

In this relation work/friendship I should include the other members of the group MODO: Maite, Carlos, Juan Ramón, Nacho, Socorro, ... who I want thank the support, advices and treatment received from the first day. I would also like to give thanks to Alejandro Sancho for supervising my first steps in this stage and because part of this thesis has emerged from his ideas.

I wish to include in these acknowledgements the Spanish Ministry of Science and Innovation for the funding of my doctoral education through the scholarship program FPI. This funding has also made possible my internship in Nottingham with the lecturer Natalio Krasnogor. I want to give thanks to him for offering me the opportunity of working with him and with other excellent researchers as Carlos García, Claudio Lima and Jamie Twycross.

From a personal point of view, there are also many people that I would like to show appreciation for the support and help received. I could not forget Fran, Javi, Fernando, Nacho, Alex, Eli and more friends since smiles, beers, parties, drinks,... also help me in doing this thesis.

Of course, during this time I have always had the support and affection of my sisters Luisa and Loli and my brother Andrés, who without any doubt

deserve an especial place in these acknowledgements. I cannot forget that in these years (and in many others) I have used the experience and advices of my brother to pave my way. For this reason I wish to give thanks to him again.

Finally, I would like to especially thank my parents for all the effort and sacrifice they have done and for the unconditional support they have offered me in every moment. I should thank them many things so I would like to dedicate this dissertation to them.

This thesis has been supported by the Andalusian Government under project TIC-02970, by the Spanish Ministry of Science and Innovation though projects TIN2005-08404-C04-01 and TIN2008-01948, and by the FPI scholarship BES-2006-13524 from this last institution.

Resumen

Este resumen contiene una versión en español de los capítulos 1 (introducción) y 7 (conclusiones), y ha sido incluido para cumplir con los requerimientos necesarios para poder optar a la mención de *Doctorado Europeo*.

Introducción

Los Sistemas Inteligentes intentan imitar el comportamiento de los seres humanos. Muchos de los problemas reales a los que se enfrenta el hombre consisten en encontrar una solución que maximice o minimice una determinada medida objetivo cuyo valor depende de ciertas variables de decisión. Este tipo de problemas pueden encontrarse en áreas tan importantes como Economía, Industria, Comercio, Logística, Bioinformática o Telecomunicaciones entre otras.

Disponer de sistemas de ayuda a la decisión o de herramientas para abordar la resolución de estos problemas de una forma eficiente y efectiva es de suma importancia. Por ejemplo, una buena distribución de una línea de producción puede incrementar de forma drástica la productividad de una industria; mejorar planificación del tráfico aéreo en los aeropuertos puede llevar a un ahorro significativo en los costes de las compañías aéreas o a una importante reducción del tiempo de espera de los clientes; la optimización del uso de los recursos sanitarios podría hacer posible el tratamiento de pacientes con enfermedades tan graves como el cáncer en un tiempo menor, etc.

Desafortunadamente, encontrar la mejor solución para muchos de estos problemas suele ser una tarea bastante compleja. Dicha complejidad se puede deber principalmente a cuatro razones: 1) el problema es NP-Duro y por lo tanto no se conoce ningún algoritmo que pueda resolverlo de forma óptima usando un tiempo y una cantidad de memoria polinomial con re-

specto al número de variables de decisión, 2) su dimensión es muy grande, 3) la modelización del problema o el conocimiento que se tiene sobre este es impreciso o vago, o bien 4) la evaluación de las soluciones es muy costosa.

En muchas situaciones es posible que no dispongamos del tiempo y/o los recursos computacionales que se requieren para encontrar la mejor solución global, o bien, puede ocurrir que el modelo con el que trabajamos esté definido de una forma vaga y por lo tanto, no tiene sentido resolver de forma óptima un problema impreciso. En estos casos, *satisfacer es que mejor que optimizar*, por lo que se hace necesario contar con herramientas que proporcionen soluciones suficientemente rápido, suficientemente buenas y suficientemente baratas. Dentro de este tipo de herramientas, sin duda alguna, las metaheurísticas han alcanzado un alto prestigio como demuestra el amplio número de revistas científicas, libros y conferencias dedicadas a este tópico. A esto hay que añadir que estas técnicas han sido empleadas con éxito en multitud de áreas y que se dispone de una gran cantidad de software tanto para su diseño y desarrollo, como para su utilización.

Sin embargo, la aplicación práctica de las metaheurísticas aún presenta ciertas dificultades que todavía no han sido resueltas. Algunas de estas dificultades son las siguientes:

- Dada una instancia de un problema de optimización, generalmente es imposible predecir cual es la mejor metaheurística para resolverlo. Esto se conoce como el Problema de la Selección de Algoritmos [131].

- Aunque conozcamos un buen algoritmo para un problema, éste puede ser ineficiente para una instancia específica.

- El funcionamiento de las metaheurísticas depende de ciertos parámetros cuyo ajuste es complejo, hasta el punto de que el tiempo necesario para implementar el método es menor que el tiempo que hay que invertir para lograr un ajuste de parámetros que permita obtener el mejor rendimiento posible.

- La definición de algunos problemas de optimización o de los requerimientos de algunos usuarios pueden presentar incertidumbre y/o imprecisión. Las metaheurísticas, en su forma básica, no han sido diseñadas para tratar con estos aspectos.

Una forma relativamente intuitiva de reducir el impacto de estas dificultades consiste en combinar diferentes heurísticas que tengan fortalezas y debilidades complementarias para crear una sinergia entre ellas y, de esta manera, construir mejores métodos de resolución de problemas. Otra posibilidad sería hibridizar las metaheurísticas con otras técnicas de la Inteligencia Artificial, como es la Soft Computing, para poder tratar la incertidumbre y la imprecisión de una forma más efectiva. El éxito de esta clase de algoritmos, llamados metaheurísticas híbridas, es notorio como prueban su creciente frecuencia de publicación y el hecho de que muchos de los mejores resultados que podemos encontrar en la literatura, tanto para problemas prácticos como académicos, han sido obtenidos por este tipo de métodos.

Dentro de las diferentes formas de hibridación, las estrategias cooperativas constituyen una de las alternativas más prometedoras. Estos métodos consisten en un conjunto de agentes cooperativos y autónomos que se ejecutan simultáneamente intercambiando información entre ellos. Diversos estudios muestran que dicha cooperación conduce a estrategias más eficientes y efectivas que sus contrapartes secuencial e independiente (no hay intercambio de información). Aparte de esto, su estructura permite una sencilla paralelización, lo que facilita el uso de la gran cantidad de recursos computacionales que hoy en día proporcionan grid y cloud computing. Estas estrategias pueden ser centralizadas, si el envío y recepción de información que los agentes llevan a cabo se realiza a través de un coordinador central que además controla su comportamiento, o bien, descentralizadas, si la información se intercambia directamente entre ellos.

Otro aspecto interesante y raramente abordado en metaheurísticas es el de como resolver un conjunto de instancias, ya que la mayoría de los estudios que se han realizado en este campo están orientados a resolver una única instancia en cada momento, sin tener en cuenta las demás. Por ejemplo, no es complicado imaginarse un servidor que recibe peticiones para resolver instancias de un mismo problema de optimización. En esta situación, ¿qué estrategia debería utilizarse para resolver eficientemente todas las instancias? Un modo sencillo de proceder en este caso sería elegir un algoritmo particular y procesar las instancias una a una. Sin embargo, operando de esta manera, se desaprovecharía la valiosa información que se obtiene durante y después de la resolución. A esto habría que añadir que en este escenario se agravarían las dificultades comentadas anteriormente. Pensemos

que podría ser necesario resolver instancias de diferentes características, por lo que el funcionamiento del algoritmo seleccionado puede ser bueno para algunas pero inaceptablemente malo para otras. La falta de herramientas a medida para este tipo de situaciones hace recomendable una mayor investigación en este tema. La aplicación de estrategias cooperativas puede ser una opción más que interesante ya que estos métodos suelen mostrar un comportamiento robusto cuando son aplicadas sobre un conjunto de instancias de un mismo problema.

Las estrategias cooperativas constituyen el tema central del proyecto de investigación co-Heurifuzzy (TIN2005-08404-C04-01), marco de trabajo de esta tesis junto con los proyectos TIC-02970, financiado por el gobierno de la comunidad autónoma andaluza, y TIN2008-01948, del Ministerio de Ciencia e Innovación español. El objetivo global de estos proyectos es el estudio, diseño y desarrollo de metaheurísticas basadas en técnicas de Soft Computing para resolver problemas difíciles en el contexto de los Sistemas Inteligentes.

Dentro de este contexto global, en esta tesis nos centraremos en los siguientes objetivos:

1. Realizar un estudio en profundidad de las estrategias cooperativas centralizadas.

2. Investigar nuevas áreas de aplicación para este tipo de métodos.

3. Validar de forma apropiada el funcionamiento de estas metaheurísticas con respecto a algoritmos del estado del arte.

4. Desarrollar y analizar estrategias cooperativas que puedan gestionar la resolución de un conjunto de instancias de instancias de una manera efectiva y eficiente.

La descripción de las tareas realizadas para alcanzar los objetivos de esta tesis se han estructurado en 5 capítulos que se describen a continuación. Los capítulos 2 y 3 proporcionan los conocimientos básicos para comprender el área de investigación. En el primero de estos capítulos comenzaremos dando conceptos básicos sobre problemas de optimización y metaheurísticas. Seguidamente se describirán brevemente algunos de los métodos basados en trayectorias y en poblaciones más conocidos. El siguiente capítulo tiene como fin introducir las estrategias cooperativas. Aquí, partiendo de un

concepto más amplio como es el de las Metaheurísticas Híbridas, nos centraremos en las estrategias cooperativas, su clasificación y revisaremos diversos enfoques actuales.

La primera contribución de esta tesis se describe en el capítulo 4. En el contexto de los problemas de optimización combinatoria, y usando una estrategia cooperativa centralizada desarrollada previamente, estudiaremos dos aspectos clave de estos métodos. Por una parte la composición de la estrategia, donde intentaremos responder la siguiente pregunta:

¿Cómo afecta la composición de la estrategia cooperativa a su funcionamiento? (estrategias heterogéneas (todos los agentes implementan una metaheurística diferente) vs homogéneas (todos los agentes implementan el mismo algoritmo))

y por otra parte, el esquema de cooperación. En este caso pretendemos dar respuesta a la siguiente pregunta:

¿Cómo influye el esquema de cooperación en el comportamiento de la estrategia?

El capítulo 5 está relacionado con el objetivo 2 y en él estudiamos nuevas áreas de aplicación para los métodos cooperativos centralizados. En este sentido, elegimos los Problemas de Optimización Dinámica (PODs) y en particular aquellos en los que la función objetivo cambia en función del tiempo. Las principales razones que nos llevaron a esta decisión fueron: a) el creciente interés en la resolución de este tipo de problemas debido a su cercanía a situaciones del mundo real (predicciones de mercado, predicciones meteorológicas, etc.) y b) las estrategias cooperativas centralizadas no se habían aplicado a estos problemas anteriormente. Nuestra investigación está orientada a responder las siguientes cuestiones:

¿Tiene sentido aplicar las estrategias cooperativas centralizadas a este tipo de problemas?

¿Cómo pueden adaptarse estos métodos a los PODs y qué aspectos de su estructura deben modificarse?

¿Cómo es su funcionamiento en comparación con otros métodos del estado del arte para PODs?

En el capítulo 6 presentamos una nueva estrategia cooperativa centralizada que permite resolver un conjunto de instancias tanto de forma secuencial (una a una) como simultánea (todas al mismo tiempo). La estrategia se basa en un conjunto de agentes y un proceso básico de aprendizaje que se retroalimenta de la información obtenida durante la resolución de las instancias. Los estudios realizados en este capítulo intentan contestar a las siguientes preguntas:

Dado un conjunto de instancias, ¿es mejor resolverlo de forma secuencial o simultánea?

¿Funciona mejor el mecanismo de aprendizaje cuando se resuelven las instancias de forma simultánea puesto la información disponible es más abundante?

El último capítulo está destinado a discutir las conclusiones globales de esta tesis así como las futuras líneas de investigación. Esta memoria termina con la bibliografía que se ha consultado para su preparación.

Conclusiones

Esta tesis se ha centrado en el estudio, diseño, desarrollo y aplicación de estrategias cooperativas centralizadas para problemas de optimización. Los objetivos que se propusieron fueron los siguientes:

1. Realizar un estudio en profundidad de las estrategias cooperativas centralizadas.

2. Investigar nuevas áreas de aplicación para este tipo de métodos.

3. Validar de forma apropiada el funcionamiento de estas metaheurísticas con respecto a algoritmos del estado del arte.

4. Desarrollar y analizar estrategias cooperativas que puedan gestionar

la resolución de un conjunto de instancias de instancias de una manera efectiva y eficiente.

En lo que respecta al objetivo 1, partimos de una estrategia cooperativa previamente desarrollada en la que los agentes son controlados por un coordinador central que toma decisiones en base a una regla difusa, y extendimos la investigación realizada en dos puntos clave. En primer lugar, en el contexto de problemas de optimización combinatorios, estudiamos como afecta la composición de la estrategia a su funcionamiento. Usando como caso de prueba el problema del p-hub mediano con asignación simple y sin capacidades analizamos el comportamiento del método cuando todos los solvers implementan la misma heurística (estrategia homogénea) y cuando cada uno implementa una diferente (estrategia heterogénea). Para tener una referencia del funcionamiento de tales estrategias, estas fueron comparadas con la versión individual de los distintos algoritmos de búsqueda utilizados como solvers. En lo que concierne a este último punto, vimos que mediante la cooperación homogénea basada en nuestro esquema puede reducirse notablemente el error promedio de tres metaheurísticas diferentes. También se compararon las distintas estrategias cooperativas frente al mejor método individual, el cual varía de una instancia a otra. Los resultados obtenidos mostraron que la cooperación, tanto homogénea como heterogénea, lleva a valores promedios de fitness iguales o mejores que los de la mejor metaheurística individual, en prácticamente todos los casos. El último aspecto que se analizó fue la diferencia de rendimiento entre los métodos cooperativos estudiados, donde se debe destacar que a) aquellas estrategias homogéneas cuyos agentes implementan la mejor metaheurística individual se muestran como la alternativa más efectiva, y b) cuando se compara la composición heterogénea frente a la homogénea, pudo comprobarse que la primera presenta ciertas ventajas sobre la segunda. Estos resultados fueron publicados como un capítulo de libro de la serie Studies in Computational Intelligence en [106].

El segundo punto en el que se extendió la mencionada investigación fue el esquema de cooperación. Concretamente, incorporamos a dicha estrategia una regla de control basada en búsqueda reactiva [8]. Los experimentos, realizados sobre el mismo problema, mostraron como el esquema de cooperación reactivo consigue mejores resultados que la estrategia independiente (donde los agentes no intercambian información) y que dicha estrategia

cuando usa la regla de control difusa que se empleó en el caso anterior. También probamos el funcionamiento de la estrategia cuando usa ambas reglas, difusa y reactiva, al mismo tiempo. En este caso, la mayor complejidad no se ve recompensada puesto que la regla reactiva por si sola ofrece un rendimiento igual o mejor que la combinación de ambas. Por otra parte, vimos como la regla reactiva es capaz de adaptar su comportamiento según las características de la instancia y comprobamos su efectividad para detectar el estancamiento y para llevar a cabo las estrategias diversificación. Estos resultados fueron incluidos en un capítulo de libro de las series Lecture Notes in Computer Sciences [105] y en un resumen extendido en la conferencia Learning and Intelligent Optimization 2009 (LION III) [104].

Otro de los aspectos abordados en esta tesis fue el estudio de nuevos escenarios en los que aplicar los métodos cooperativos centralizados (objetivo 2). Se eligieron como escenario los Problemas Dinámicos de Optimización (PODs) y concretamente, aquellos cuya función objetivo cambia en función del tiempo. La estrategia cooperativa desarrollada para tratar con estos problemas tenía la misma estructura que la usada anteriormente. Los agentes implementaban un algoritmo basado en trayectorias, cada uno de ellos con una configuración de parámetros diferente. Comprobamos que este tipo de métodos puede ser fácilmente adaptados a PODs mediante la incorporación de un mecanismo que controle los cambios de la función objetivo y programando el reinicio de ciertas memorias cada vez que uno de estos cambios tiene lugar. Hasta donde alcanza nuestro conocimiento, los métodos basados en trayectorias y el esquema de cooperación presentado no habían sido aplicados anteriormente a PODs. El método estudiado obtuvo resultados muy prometedores que mejoraban de forma significativa a dos algoritmos del estado del arte para estos problemas (Agents y multi-QPSO) en la mayoría de los casos de test considerados (objetivo 3). Además, se suele creer y asumir que los métodos basados en poblaciones son más adecuados parar tratar con PODs ya que supuestamente contar con un mayor número de soluciones puede llevar a un mejor rastreo de los cambios que tienen lugar en este tipo de entornos. Aunque esto todavía puede considerarse cierto, el éxito de esta estrategia cooperativa centralizada basada en métodos de trayectoria puede tener un papel importante el campo de la optimización dinámica. La revista Memetic Computing ha aceptado para su publicación un artículo en que el incluían estos resultados [60].

Para completar el cuarto objetivo, propusimos un método basado en estrategias cooperativas para resolver un conjunto de instancias con la idea de que el esfuerzo realizado y el conocimiento obtenido durante la resolución de una instancia deben usarse para la solución de otras. Nuestra estrategia puede verse como un sistema co-evolutivo donde las soluciones de diferentes instancias son mejoradas iterativamente y donde diversos operadores son gestionados dinámicamente en función de su rendimiento sobre las instancias que ya han sido resueltas. Otro aspecto importante de esta estrategia consiste en que permite operar en dos modos de funcionamiento diferentes: secuencial, en el que las instancias se resuelven una a una, y simultáneo o paralelo, en el que todas las instancias se procesan al mismo tiempo. La experimentación se llevó a cabo utilizando SAT como banco de pruebas y en ella, a parte de tres algoritmo específicos para este problema, se consideraron las siguientes cuatro configuraciones de nuestra estrategia:

- NAS: secuencial sin adaptación (aprendizaje desactivado)
- AS: secuencial con adaptación (aprendizaje activado)
- NAP: simultáneo sin adaptación (aprendizaje desactivado)
- AP: simultáneo con adaptación (aprendizaje activado)

La principal conclusión que se desprendió de esta experimentación fue que, incorporando algoritmos específicos del problema, nuestra estrategia (especialmente la versión AP) mejora a los algoritmos de propósito específico (no adaptativos) tanto en términos de eficacia (número de instancias sin resolver) como de eficiencia (evaluaciones de la función objetivo). La comparación de los modos de funcionamiento reveló que el caso simultáneo ofrece un comportamiento más robusto en términos de eficacia (AP mejor que AS, y NAP mejor que NAS) debido a un mejor uso de los recursos (evaluaciones) disponibles. A esto hay que añadir que la versión simultánea cuenta con una ventaja adicional: supera al modo secuencial puesto que resuelve un mayor número de instancias en menos tiempo debido a que, de una manera implícita, lleva a cabo la resolución de la instancias más fáciles en primer lugar, dejando las difíciles para las últimas etapas de la búsqueda. El uso de la adaptación y el mecanismo de recompensa fueron claramente beneficiosos. En particular, AS fue mejor que NAS, y AP fue mejor que NAP, puesto que requirieron de un menor número de evaluaciones para resolver

el mismo conjunto de instancias. El método desarrollado y los resultados obtenidos fueron publicados en la revista Soft Computing [107].

Antes de terminar, debemos apuntar que para llevar a cabo la experimentación de esta tesis, hicimos uso de dos herramientas desarrolladas dentro del Grupo de Investigación en Modelos de Decisión y Optimización:

- SiGMA: un sistema de ayuda a la decisión basado en optimización que proporciona un gestor de algoritmos de búsqueda, potente y dinámico, que facilita su gestión y comparación, así como el análisis de los resultados obtenidos.
- DACOS, un sistema integrado de soporte en el diseño y análisis de sistemas de optimización cooperativos y centralizados.

SiGMA se utilizó para el ajuste de los diferentes solvers que componían las estrategias, mientras que DACOS se empleó en la configuración y ajuste de los métodos cooperativos evaluados en esta tesis. El autor contribuyó a la mejora de estas herramientas y por lo tanto a su publicación en las revistas científicas Expert Systems with Applications [68] y Software: Practice and Experience [41] (sometido), respectivamente.

Abstract

This dissertation focuses on the study, design, development and application of centralised cooperative strategies for optimisation problems. Those models consist on many parallel cooperating agents, where each agent carries out a search in a solution space. Firstly, we study the most known trajectory-based and population-based metaheuristics and next, the cooperative strategies are introduced. The contributions of this thesis start analysing some aspects of the centralised cooperative strategies as the composition and the cooperation scheme. Using the Uncapacitated Single Allocation p-Hub Median Problem as test bed, we compare the performance of homogeneous and heterogeneous strategies and give some insights about the benefits of each type of composition. Using the same problem, we test a cooperation scheme based on Reactive Search ideas proposed by the author and we compare it against other techniques. The results show the better performance of the reactive scheme. Another issue tackled in this dissertation is the application of centralised cooperative strategies to Dynamic Optimisation problems, where they have not been applied before. The method is evaluated over different benchmarks obtaining a very robust performance that improves two state-of-the-art methods for these problems. Finally, a cooperative method that allow the resolution of a set of instances is presented. The strategy is based on a set of operators and a basic learning process that is fed up with the information obtained while solving several instances. The output of the learning process is an adjustment of the operators. The instances can be managed sequentially or simultaneously by the strategy. The method has been tested on different SAT instance classes and the results confirm that a) embedding problem specific algorithms into our strategy, instances can be solved faster than applying these algorithms instance by instance and b) that the simultaneous resolution of instances performs better than the sequential one.

Part I

Introduction

Chapter 1

Introduction

Intelligent Systems attempt to mimic the behaviour of human beings. Many of the real-life problems that humans need to solve consist on finding a solution that maximise or minimise a determined objective measure whose value depends on certain decision variables. These type of problems appear in such an important areas as Economy, Industry, Commerce, Logistics, Bioinformatics or Telecommunications among others.

The importance of having efficient and effective resolution or decision support tools for these problems is crucial. A better distribution of a production line can improve the productivity of industry drastically; more efficient scheduling of air traffic in airports can lead to significant financial savings for air companies or waiting time reduction for costumers; a more optimised used of healthcare resources can make possible the treatment of patients with important diseases as cancer in a shorter time, etc.

Unfortunately, finding the best possible solution for many of these problems is usually a complex task. This complexity can primarily be due to four reasons: 1) the problem is NP-hard and hence an algorithm that can obtain the global optimum using a polynomial amount of time or memory is not known, 2) its dimension is very high, 3) the modelisation of the problem or the knowledge about it is imprecise or vague, or 4) evaluating the cost of a solution is very expensive.

In many situations we may not have the time and/or the computational resources needed to find the best global solution, or for example, the model is vague so talking about the optimum of an imprecise problem makes no sense. In these cases, *satisfying is better than optimising* and hence, it is necessary to have tools that can provide 'soon-enough, good-enough and

cheap enough' solutions. Within this type of tools, without any doubt, metaheuristics have become very successful as the wide number of scientific journals, books and conferences dedicated to this topic shows. Besides this, they are applied in multiple areas and a large amount of software is available for their design, development and use.

However, the practical application of metaheuristics still presents drawbacks that have not yet been solved. Some of these drawbacks are the following:

- Given a problem instance, it is generally impossible to predict which is the best heuristics to solve it. This is known as the Algorithm Selection Problem [131].
- Although we know a good algorithm for a problem, this can be inefficient for a specific instance.
- The performance of the metaheuristics depends on certain parameters whose adjustment is complex to the point that the time needed to implement a metaheuristic is much lower than the time required to fine tune these parameters in order to obtain the best possible performance. Furthermore, such tuning is not usually suitable for others problems or even for others instances of the same problem.
- The definition of some optimisation problems or the requirements of some users can present uncertainty and/or vagueness. Metaheuristics in their basic definition are not designed to deal with these features.

A relatively intuitive way to reduce the impact of these drawbacks consists on combining different heuristics that have complementary weaknesses and strengths to create a synergy among them in such a way that better problem solvers could be obtained. Another possibility is to incorporate other techniques from Computational Intelligence as Soft Computing into metaheuristics in order to deal with uncertainty and vagueness in a more effective manner. This class of search algorithms, called *hybrid metaheuristics*, are increasingly reported and have obtained many of the best results published in the literature for both practical and academical problems.

Within the different forms of hybridization, cooperative strategies constitute one of the most promising alternatives. These methods consist on a set of self-contained cooperating agents that are run in parallel exchanging

information among them. Many studies have shown that this cooperation leads to more efficient and effective strategies if they are compared with its sequential and independent (no information exchange) counterparts. Furthermore, its structure makes really easy its parallelisation which facilitates the using of the big amount of computational resources that nowadays grid and cloud computing provide. These strategies can be centralised, if the sending and reception of information carried out by the cooperating agents is done through a central coordinator that controls its behaviour, or decentralised, if the information is exchanged directly between them.

It is also interesting to know that a problem hardly tackled in metaheuristics is how to solve a set of instances, since the majority of the studies done in this field are oriented to solve one instance at a time, independently of the other ones. For example, it is not hard to imagine a server that receives requests to solve instances of the same optimisation problem. Under this situation, which strategy should be used to efficiently solve all the instances?. A naive mode of operation would be to choose a particular algorithm and to solve the instances one by one. However, working in this way, valuable information obtained during and after the resolution process of every instance is discarded. Besides, the drawbacks we pointed out are exacerbated. For example, it could be necessary to solve instances of different characteristics, thus the performance of the selected algorithm can be excellent for some instances but unacceptably bad for others. The lack of specialized tools for this type of situations makes advisable the research on this topic. The application of cooperative strategies can be an interesting option inasmuch they usually show a robust behaviour over a wide set of instances of the same problem.

Cooperative strategies represent the central issue of the research project co-HeuriFuzzy (TIN2005-08404-C04-01) which is the framework of this thesis together with the projects TIC-02970, financed by the Andalusian Regional Government, and TIN2008-01948, from the Spanish Ministry of Science and Innovation. The global objective of these projects is the study, design and development of cooperative metaheuristics based on Soft Computing techniques to solve hard problems in the context of the Intelligent Systems.

Within this global context, in this dissertation we focus on the next objectives:

1. Make an in-depth study of centralised cooperative strategies.
2. Research on new application areas for this type of methods.
3. Validate the performance of these metaheuristics with respect to state-of-the-art algorithm in a proper manner.
4. Develop and analyse cooperative strategies that can manage the resolution of a set of instances in a effective and efficient way.

The description of the tasks done to achieve the objectives of this thesis has been structured in 5 chapters that are described below. Chapters 2 and 3 provide the background needed to understand the research area. In the first of these chapters, we start giving the basic concepts of optimisation problems and metaheuristics. Then, some of the most known trajectory-based and population-based methods are briefly described. Next chapter is devoted to introduce the cooperative strategies. Departing from the wider concept of Hybrid Metaheuristics we will focus on cooperative strategies, their classification and a review of the current approaches.

The first contribution of this dissertation is described in chapter 4. In the context of combinatorial optimisation problems, and using a centralised cooperative strategy previously developed, we study two key aspects of these algorithms. On one hand, the composition of the strategy, where we try to answer the following question:

How does the composition of a centralised cooperative strategy affect its performance? (heterogeneous (all agents implement a different metaheuristic) vs homogeneous strategies(all agents implement the same algorithm))

and on the other hand, the cooperation scheme. In this case we address the next question:

How does the scheme of cooperation influence the behaviour of the strategy?

Chapter 5 is related to objective 2 and here, we study new application areas for centralised cooperative methods. In this sense, we chose Dynamic Optimisation Problems (DOPs), and in particular those in which the objective function changes over the time. Two main reasons lead us to this

election: a) there is a growing interest on the resolution of these problems due to its closeness to real-world situations (trade market predictions, meteorological forecast, etc.) and b) centralised cooperative strategies have not been applied to these problems before. Our research is oriented to answer the following:

Does it make sense to apply centralised cooperative strategies to this type of problems?

How can these methods be adapted to DOPs and what aspects of their structure should be modify?

How is their performance compared to other state-of-the-art methods for DOPs?

In chapter 6 we present a new centralised cooperative strategy that allows to solve a set of instances both sequentially (one by one) and simultaneously (all instance at the same time). The strategy is based on a set of agents and a basic learning process that is fed back with the information obtained while solving the instances. The output of the learning process is an adjustment of the operators used by the agents. The analysis done in this case is oriented to address the questions:

Given a set of instances, is it better to solve it in a sequential or simultaneous way?

Does the learning mechanism perform better when the instances are solved simultaneously since the information available is richer?

The last chapter is devoted to discuss the global conclusions of this thesis as well as the future lines of research. The dissertation ends with the bibliography consulted for its preparation.

Part II

Preliminaries

Chapter 2

Soft Computing and Metaheuristics

In this chapter we will describe Soft Computing following the work presented by Verdegay, Bonissone and Yager in [147] where besides from reviewing which are its classic components, we will see that Metaheuristics are a fundamental part of this area. Next part of the chapter is focused on these optimisation methods. Firstly, some basic definitions related to optimisation will be introduced in order to unify terms that will be used in this dissertation. Afterwards some of the most known Metaheuristics will be presented, we will describe their essential characteristics and provide some references to guide the reader. Finally, a summary of the chapter will be given.

2.1 Soft Computing

The necessity of solving some problems optimally or determining the best solution among the available ones have lead to the scientific community to develop and study theories as well as propose suitable methodologies for the field in which the question arises. Within this general frame an important part is occupied by the optimisation problems: those whose resolution consists on finding the maximum or minimum value that a given function can take in a previously defined set. Everything relating to these problems is framed within Mathematic Programming area. This area embrace a wide variety of models (linear and non linear cases, randomness, one or several

decisors, etc.) although, without any doubt, the single objective linear case, model studied by Linear Programming, is the one that has focused the most attention and has shown the most successful practical applications.

Some Mathematical Programming problems can have elements that present vagueness. To deal with these problem, fuzzy optimisation methods were developed, being one of the most successful areas within the fuzzy context both in practical and theoretical sense. However, despite the wide variety of practical situations that can be tackled by fuzzy optimisation methods, there are some scenarios in which these techniques can not solve the arisen problems.

The big computational power available nowadays together with the fact that some problems with a great practical value (scheduling, planning, routing, facility location, cutting-packing, etc.) can not be solved optimally, have lead to an increasing use of heuristics techniques in those situations where the exact algorithms can not give an answer using a reasonable amount of time or memory. Following the principle "satisfy is better than optimise" a wide variety of highly efficient and effective heuristic methods have been proposed in the literature. Some of these heuristics are Tabu Search, Simulated Annealing, GRASP (Greedy Randomized Adaptive Search), Genetic Algorithms and other more recent as Memetic Algorithms, Estimation Distribution Algorithms, Variable Neighbourhood Search, Ant Colony Optimisation, Scatter Search, ... This list shows the big interest in this field and the lack of a theoretical frame where locating, relating and comparing these algorithms.

Lofti A. Zadeh presented in 1965 [157] the idea of fuzzy set, where the membership of an element to set is given by a value in the interval [0,1] instead of the values {0,1} as the classic set theory establishes. From this moment, several applications and developments based on this basic idea have arisen, having a big impact in many areas as industry, electronic devices, transport, etc.

Zadeh was also who proposed the first definition of Soft Computing in 1994 [158], although the idea of establishing the area of Soft Computing has already appeared in 1990 (see [159]). Until that time, the concepts that involved Soft Computing were dealt in a isolated way with an indication of using fuzzy methodologies. The definition proposed by Zadeh was the following:

> Basically, soft computing is not a homogeneous body of concepts and techniques. Rather, it is a partnership of distinct methods that in one way or another conform to its guiding principle. At this juncture, the dominant aim of soft computing is to exploit the tolerance for imprecision and uncertainty to achieve tractability, robustness and low solutions cost. The principal constituents of soft computing are fuzzy logic, neurocomputing, and probabilistic reasoning, with the latter subsuming genetic algorithms, belief networks, chaotic systems, and parts of learning theory. In the partnership of fuzzy logic, neurocomputing, and probabilistic reasoning, fuzzy logic is mainly concerned with imprecision and approximate reasoning; neurocomputing with learning and curve-fitting; and probabilistic reasoning with uncertainty and belief propagation.

In this way, Zadeh defined Soft Computing by extension, by means of different techniques and concepts developed to tackle problems that present elements with imprecision, uncertainty or that are difficult to categorize.

Other authors tried to give a more precise definitions though the attempts were not very successful. For example, in [92], taking into account the difficulty of giving an exact and agreed definition for Soft Computing, Li et al. described this area by its characteristics:

> Every computing process that purposely includes imprecision into the calculation on one or more levels and allows this imprecision either to change (decrease) the granularity of the problem, or to "soften" the goal of optimization at some stage, is defined as to belonging to the field of soft computing.

More recently, Verdegay, Yager and Bonissone [147] have presented a more precise and illustrative definition of what is currently Soft Computing. This is quoted below:

> The viewpoint that we will consider here (and which we will adopt in the future) is another way of defining soft computing, whereby it is considered to be the antithesis of what we might call hard computing. This viewpoint is consistent with the one

> in [158, 159]. Soft computing could therefore be seen as a series of techniques and methods so that real practical situations could be dealt with in the same way as humans deal with them, i.e. on the basis of intelligence, common sense, consideration of analogies, approaches, etc. In this sense, soft computing is a family of problem-resolution methods headed by approximate reasoning and functional and optimization approximation methods, including search methods. Soft computing is therefore in the theoretical basis for the area of intelligent systems.

From the perspective of this definition, soft computing can be extended to in a second level of components among which probabilistic reasoning, fuzzy logic and fuzzy sets, neural networks and genetic algorithms (GA) was stood out as the most important ones from the beginning [19]. The popularity of GA's together with the wide number of extensions and versions proposed, transformed the fourth component of the second-level in the so called Evolutionary Algorithms (EA).

However, Evolutionary Algorithms are just a class of metaheuristics as Hill Climbing, Tabu Search, Simulated Annealing, Variable Neighbourhood Search, Greedy Randomized Adaptive Search Procedure, Scatter Search or Ant Colony Optimisation, among others, can be. These heuristics also fulfil the principle "it is better to satisfy than to optimise" since they provide solutions with a reasonable quality for the users or the decisors, whereby they are perfectly adapted to Zadeh's sentence [158] : "... in contrast to traditional hard computing, soft computing exploits the tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, low solution-cost, and better rapport with reality". In this manner, instead Evolutionary Algorithms, heuristic algorithms or better Metaheuristics can be considered as the fourth component of Soft Computing.

The term heuristic come from the Greek word "heuriskein" whose meaning is associated to the concept of finding and is related to the famous sentence exclaimed by Arquimedes "eureka!". As for the term Metaheuristic, this was firstly used by Glover in 1986 [62] and results from adding the prefix *meta* to the word heuristic. This prefix indicates that are methods in a higher level of abstraction. Metaheuristics arose with the idea of extracting the best parts of different successful heuristics to create generic methods that could be applied to a larger number of problems and contexts. There

is no agreed and formal definition of Metaheuristic, although the next two, in our opinion, give a clear idea of what these methods are:

a) "An iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space" [119]

b) "An iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method" [150]

Taking into account these definitions, it is clear that Evolutionary Algorithms are included within the more general concept of Metaheuristic, and therefore, they can be considered as one of the second-level components of Soft Computing. In this way, Soft Computing embrace a wider area which makes easier the development of new outlines, theoretical and practical methodologies and frameworks that allow a better processing of the uncertainty and imprecision. According to this, we could say that Soft Computing, in its second level, has as main components Probabilistic Reasoning, Multivalued and Fuzzy Logics, Neural Networks and Metaheuristics.

The methodologies that integrate Soft Computing should be seen as the result of the cooperation, association, complementarity or hybridization of the components stated above. Upon considering the inclusion of Metaheuristics as one of the elements of Soft Computing, it is important to analyse the new theoretical and practical aspects derived from here. In this sense, we should take into account that there exist a large variety of Metaheuristics that can be classified in the next groups:

1. Evolutionary methods in which a population of solutions is evolved by simulating the natural processes that take place in the evolution of species.

2. Relaxation methods where the original model to solve is adapted by the algorithm to facilitate its resolution.

3. Metaheuristics that base their working on the exploitation of the neighbourhood structure of the solutions which compose the search space.

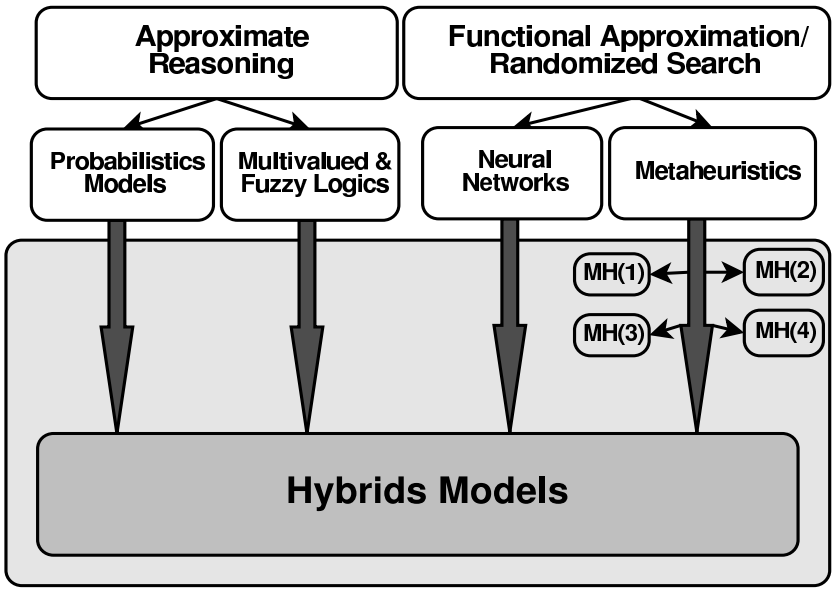


Figure 2.1: Soft Computing components

4. Metaheuristics created combining two or more of the previous methods or derived from them.

Although there is no a universally accepted classification for metaheuristics, the taxonomy used above counts with a big consensus. Naming these groups as MH(1),. . . ,MH(4), Soft Computing and its main components can be represented as it is shown in Figure 2.1.

2.2 Metaheuristics

Once we have describe what Soft Computing consists on and which their elements are, we will only focus on the component in which this thesis is framed, that is, Metaheuristics. In the next part of this chapter some basic concepts about optimisation besides some of the most known approaches will be seen.

2.2.1 Basic Concepts

An instance of a optimisation problem P is define as $(\mathcal{S}, f)$ and is composed by

- a set of decision variables $X = \{x_1, \ldots, x_n\}$
- the domains of the variables $D_1, \ldots, D_n$
- a set of constraints among variables
- a objective function $f : D_1 \times \ldots \times D_n \rightarrow R^+$

$\mathcal{S} = \{s = (x_1, v_1), ..., (x_n, v_n)\}$ with $v_i \in D_i$, $i = 1, \ldots, n$, is the set of feasible assignments and is composed by those s that fulfil all the constrains. $\mathcal{S}$ is called search space.

To solve the problem is necessary to find a solution that minimise/maximise the value of the objective function. From now on, we will suppose, without lost of generality, that we are minimising. Formally, solving a optimisation problem consists on finding a solution $s^* \in \mathcal{S}$ such that $f(s^*) \leq f(w), \forall w \in \mathcal{S}$. The solution s^* is named the optimum of $(\mathcal{S}, f)$.

Now, let $(\mathcal{S}, f)$ be an instance of a optimisation problem. The neighbourhood structure $\mathcal{N}$ is defined as a function $\mathcal{N} : \mathcal{S} \rightarrow 2^{\mathcal{S}}$, which determines for each solution $s \in \mathcal{S}$ the set $\mathcal{N}(s) \subseteq \mathcal{S}$ of solutions "close" (in some sense) to s.

The set $\mathcal{N}(s)$ is called the neighbourhood of the solution s. Examples of neighbourhoods rise for instance, if a distance function is used:

$$dist : \mathcal{S} \times \mathcal{S} \rightarrow R$$

The neighbourhood of the solution s can be defined as:

$$\mathcal{N}(s) = \{y \in \mathcal{S} \mid dist(s, y) \leq \epsilon\}$$

The following definition can be also used:

$$N(s) = \{y \in \mathcal{S} \mid \rho(y, s) = True\}$$

where $\rho(a, b)$ is a boolean predicate.

In general, the solutions y are obtained from the application of an operator $\mathcal{O}$, which is usually denominated "Move" or "Pivot", that modifies in some sense the current solution s to obtain new solutions. This operator usually includes a random process, and therefore successive applications of $\mathcal{O}(s)$ allow to obtain different solutions y.

Algorithm 1 Local Search.

```
procedure LocalSearch
  s ← GenerateInitalSolution()
  repeat
    s' ← s
    s ← Improve(s')
  until f(s) < f(s')
  return s
```

Then, it makes sense to talk about the neighbourhood of s respect to $\mathcal{O}$ and can be defined as:

$$\mathcal{N}_{\mathcal{O}}(s) \ = \{\hat{s}_i \mid \hat{s}_i = \mathcal{O}_i(s)\}$$

where $\mathcal{O}_i(s)$ stands the i-*th* application of $\mathcal{O}$ to s.

Once the neighbourhood of a solution s is properly defined as $\mathcal{N}(s)$, and given a initial solution s_0, it is possible to establish an iterative improvement scheme that starts finding a solution $s_1 \in N(s_0)$ such that s_1 fulfils a determined condition. For instance, that one which improves the fitness of s_0.

Subsequently, s_1 becomes the current solution and the process is repeated until a stopping criterion is met or it is not possible to obtain a solution $s_{k+1} \in N(s_k)$ that satisfies the established conditions. In this moment, we say that s_k is a local optimum.

This class of methods also receives the name of *local search* [1] and in Algorithm 1 its elemental pseudocode is shown. The function *improve(x)* returns, if it possible, a new neighbouring solution y such that y is better than s. Otherwise, it returns the current solution, that corresponds with a local minimum. This basic scheme is named Hill Climbing.

There exist, at least, two basic strategies to implement the procedure *Improve(x)*: the strategy *First*, which returns the first solution of the neighbourhood that improves the fitness of x, and the strategy *Best*, that explores the neighbourhood in a exhaustive way and returns the best solution found. Both strategies, also call *pivot rules*, finishes when a local optimum is reached whereby the final quality of the solutions is strongly influenced by the definitions of $\mathcal{S}$, f and $\mathcal{N}$.

Algorithm 2 Multi-Start Local Search.

```
procedure MultiStartLocalSearch
  s ← GenerateInitalSolution()
  restart ← 0
  while restart < MAX do
    repeat
      s' ← s
      s ← Improve(s')
    until f(s) < f(s')
    if f(s) < f(best) then
      best ← s
    end if
    s ← GenerateInitalSolution()
    restart ← restart + 1
  end while
  return best
```

Despite its simplicity, these methods show an important drawback: they usually get trapped in local minima. As a consequence it is necessary to extend these algorithms by means of additional mechanisms that allow to face such situation.

The simplest manner to extend this scheme is presented in Algorithm 2, where simply the search is restarted from a new solution, when the current one can not be improved.

This is known as Multi-Start [101, 102] and establishes guidelines to restart in a intelligent way the descending searches. The Multi-Start search procedures do various monotonous searches beginning from different initial solutions. One of the most simple possibilities consists on generate a sample of initial or starting solutions. This is equivalent to generate a starting random new solution each time the search is stagnated in the region close to a local optimum. We can talk about two phases. In the first one, a solution is constructed whereas the second one tries to improve it by means of a local search method. The initial solutions can be generated by a random process or by more sophisticated methods that consider the problem properties to obtain good quality solution.

Another mechanism to avoid local optima consists on include the basic scheme of a local search in a higher level, that is, in the metaheuristics.

As we said in Section 2.1, there are distinct ways of classifying the metaheuristics, among which the following ones [17] can be found: considering the origin of the method, we can talk about "bioinspired" algorithms (genetic algorithms, ant colony optimisation, particle swarm optimisation) vs. "non bioinspired"; another possibility is to categorize metaheuristics by the utilisation or not of memory; or in function of the use of a static or dynamic objective function. One interesting classification consist on differentiated those methods that keep a unique solution (trajectory-based metaheuristics), that is, those methods that move along the search space of the problem modifying iteratively the starting solutions, from those ones that keep a set or population of solutions (population-based metaheuristics) [31].

The next part of the chapter is devoted to describe some of the most known approaches. Making an exhaustive compilation of these methods is out of the scope of this work so only essential characteristics of these ones are described. We provide references that can guide the interested reader.

2.2.2 Simulated Annealing

Simulated Annealing [26, 88] is one of the most ancient metaheuristics and was one of the first algorithms that contained a explicit strategy to scape from local minima basing on the fundamental idea of allowing non improvement movements. The acceptance probability of this type of moves is decreasing along the search, and depends on the fitness of the current solution $f(s)$ and a parameter control that modulates which proportion of bad solutions is allowed. The general working scheme is described in Algorithm 3.

The algorithm starts generating a initial solution (which can be random or also constructed heuristically) and setting up a temperature parameter (T) to a high initial temperature (T_0). In each iteration, a solution $s\prime$ from $\mathcal{N}(s)$ is randomly chosen. This is accepted as new current solution in function of its objective value $f(s\prime)$, the fitness of s ($f(s)$) and the current temperature T. In this way, $s\prime$ replace s every time its objective value is better, otherwise it will do it with a certain probability that depends on the temperature and $f(s\prime) - f(s)$. This probability is usually computed making

Algorithm 3 Simulated Annealing.

```
procedure SimulatedAnnealing
  s ← GenerateInitalSolution()
  T ← T_0
  while not stopping condition do
    s' ← RandomChose(N(s))
    if f(s') < f(s) then
      s ← s'
      if f(s) < f(best) then
        best ← s
      end if
    else
      prob_accept ← p(T, s', s)
      s ← AcceptanceCriterion(s', s, prob_accept)
    end if
    Update T
  end while
  return best
```

use of the Bolztman's distribution ($exp(-\frac{f(s\prime)-f(s)}{T})$) [17].

The temperature T is reduced as search keeps on. In this way, the probability of allowing non improvement movements is high at the beginning and decreasing gradually thus the strategy converges towards a Hill Climbing method. This process is analogous to the annealing of the metals or the crystal, that leads them to a low energy configuration if they are annealed with a proper planning. The choice of the annealing policy is a very important aspect for the algorithm performance. Although it has been shown theoretically that certain annealing policies drive to a global optimum, the number of iterations needed to achieve it has an exponential order in the size of the solution space. For this reason, in practical applications more efficient policies that does not warrant this convergence are used.

2.2.3 Tabu Search

The metaheuristic Tabu Search [63, 65], proposed by Glover, is one of the most used trajectory-based methods. Its working is divided on 3 phases: *preliminary search*, *intensification* and *diversification*. In the first phase, this heuristics explores the neighbourhood of the current solution ($\mathcal{N}(s)$), and move to the best solution $s\prime$ of that neighbourhood even if it is worse

Algorithm 4 Tabu Search.

procedure *TabuSearch*()
  $s \leftarrow$ **GenerteInitialSolution()**
  Initialise tabu lists
  while not stopping condition do
    PermittedSet(s) $\leftarrow$ {$s' \in \mathcal{N}(s)$ | **tabuConditions**(s')=$false$ **OR aspirationCriteria**(s')=$true$}
    $s \leftarrow$ **ChooseBest(PermittedSet**(s))
    if $f(s) < f(best)$ **then**
      $best \leftarrow s$
    end if
    Update aspiration criteria and tabu lists
  end while
  return *best*

than s.

This type of movements within the search space can lead to the occurrence of cycles between two solutions. To avoid this situation, Tabu Search forbids the last l movements. This anti-cycles mechanism is implemented by keeping a short-term memory, call "tabu list", where the inverse of the last l movements are stored (the elements are inserted and removed in a FIFO order).

In the *intensification* stage, the tabu list is cleaning and the search starts from the best solution found, proceeding as in the preliminary search for a certain number of movements. When the *diversification* phase takes place, the tabu list is cleaning again and the l more frequent movements done until that moment are placed on it. Then some preliminary search steps are done starting from a random solutions. In this way, Tabu search alternates more exhaustive optimisations in the promising regions found by the *intensification* phase with the exploration of new regions in the *diversification* stages.

The handling of list that store complete solutions is inefficient, so the tabu list usually keep solution attributes. The attributes can be components of the solutions, movements or differences between two solutions. When more than one feature is considered, each of them is stored in a different tabu list. These sets of attributes together with the tabu list allow to define the permitted set, that is, those solutions that fulfil the *tabu conditions*.

But the storing of solution features has an associated drawback, the lost of information. Since the same attribute can be found in more than one points of the search space, the fact of forbidding one of them can avoid to search unexplored regions. To overcome this problem, some *aspiration criteria* can be establish to allow the inclusion of certain solutions in the permitted set even if they are forbidden by the tabu conditions. Allowing solutions that are better than the best solution found is a common aspiration criterion. In general tabu movements will be permitted if the aspiration criterium determines that can be profitable. A basic example of Tabu Search in displayed in Algorithm 4.

2.2.4 Greedy Randomized Adaptive Search Procedure

Greedy Randomized Adaptive Search Procedures (GRASP) can be considered an hybrid between constructive and local search heuristics as can be seen in Algorithm 5. The working of this method alternates the construction of a solution with an improvement process of that solution.

The construction of solutions does not follow a deterministic procedure but is adapted dynamically and has a randomized component. Assuming that a solution s can be build joining a subset of elements from the set "solution components", the solution is constructed iteratively adding element by element. The next "piece" to add is chosen in the next way: the possible components are ordered by a heuristic criterion that assigns them a determine score taking into account the benefit obtained when they are added to s. Then, the element to be incorporated is randomly chosen from a list constituted by the best scored components, call Restricted Candidate List (CRL). It is important to note that the benefit is inferred in a blind manner, since it only considers the profit obtained by adding the candidate to the solution at that moment. Beside this, the heuristics scores are adapted dynamically depending on the options available. A possible heuristic of this type is the election of the most cheapest candidate, being the score of an element the cost of adding it to the partial solution.

The influence of the heuristic in the search is given by the length of the CRL. A list of length n makes the generation of the solution completely random whereas a list of length 1 leads to a pure greedy construction. The adjustment of this parameter is crucial since it determines to what extent the search space is explored and therefore, the performance of the method. The

Algorithm 5 Greedy Randomized Adaptive Search Procedure (GRASP).

```
procedure GRASP()
  while not stopping condition do
    s ← CreateGreedySolution()
    LocalSearch(s)
    if f(s) < f(best) then
      best ← s
    end if
  end while
  return best
procedure CreateGreedySolution()
  s ← ∅
  l ← CalculateLengthCadidateList()
  while ! solution constructed do
    LRC_l ← GenerateCandidateList(s)
    x ← RandomChose(LRC_l)
    Add element x to s
  end while
```

parameter can be set to a fixed value or adapted by a certain mechanism.

The second phase is just a local optimisation process where different trajectory-based algorithms can be used (Hill Climbing, Simulated Annealing, Tabu Search, etc).

To obtain a good performance from a GRASP algorithm, the mechanism to construct the solutions should permit the exploration of promising regions of the search space, which can be achieved by choosing a proper heuristic and a suitable length for the CRL. Furthermore, another important condition is the location of the solutions generated in the construction stage in basins close to local optima. This can be accomplish by combining constructive heuristics and local searches with complementary features.

Although GRASP can be outperformed by more complex methods, it simplicity makes them very fast algorithms that can find good solutions in reasonable times which can be interesting in many practical situations.

2.2.5 Variable Neighbourhood Search

The Variable Neighbourhood Search, proposed by Nenad Mladenovic and Pierre Hansen [113], is based on the idea of changing systematically the

Algorithm 6 Variable Neighbourhood Search (VNS).

```
procedure VNS()
  N_k, k = 1, ..., k_max are the chosen neighbourhood structures
  s ← GenerateInitialSolution()
  while not stopping condition do
    k ← 1
    while k < k_max do
      s' ← RandomChose(N_k(s))
      s'' ← LocalSearch(s')

      if f(s'') < f(s) then
        s ← s''
        if f(s) < f(best) then
          best ← s
        end if
        k ← 1
      else
        k ← k + 1
      end if
    end while
  end while
  return best
```

neighbourhood structure in order to escape from local minima [72, 73]. The principles of this idea are the followings:

1. A local optimum with respect to a neighbourhood structure can not be so for another.

2. A global optimum is a local mininum for all the possible structures.

3. The local minima with the same or a different neighbourhood structure are relatively close in many problems.

Last principle is based on empirical evidences and implies that the local optima provide information about the location of the global minimum because they probably share common characteristics. These facts suggest that the using of different neighbourhood operators or structures for local search can be a good strategy to solve optimisation problems.

The basic form of this metaheuristic is given in Algorithm 6. To apply VNS is necessary to define the different neighbourhood structures that VNS

will work with. A common option is to choose nested neighbourhoods, that is, $\mathcal{N}_1 \subseteq \mathcal{N}_2 \subseteq \ldots \mathcal{N}_{k_{max}}$. Within the working of the method we can find three phases. Firstly, the shaking stage where a solution $s\prime$ from the k-th neighbourhood of the current solution is randomly picked up. In the second phase, a local-search algorithm is applied taking $s\prime$ as starting solution. The solution obtained from this local optimisation process is compared against the current solution s. If the new solution is better, then it replaces s and the search is restarted with k=1. Otherwise, k is incremented and a new shaking phase with this new neighbourhood structure is done. The local search method can use any neighbourhood structure without being restricted to $\mathcal{N}_k, k = 1, ..., k_{max}$.

The aim of the shaking movement is to take out the local search from the local minimum and locate it in a good starting point. This point should be in the basin of attraction of another local optimum but it is important that both optimums are relatively close because good solutions usually have some common attributes with the current local minimum. Big shaking movements would convert the search in a simple multi-start method that does not take into account the information obtaining during the optimisation process.

VNS carries out a progressive diversification that achieves by changing the neighbourhood structure systematical. The success of these systematical changes is due to the fact that different neighbourhood operators provide different fitness landscapes for the search, whereby an adverse region for a structure can not be for other one, as it happens with the local minima.

A well known variant of VNS is Variable Neighbourhood Descent. This method is based on principle 1 and consists on a local search where the neighbourhood structure of the current solution is changed if a better neighbour is not found. Another version of VNS is the general VNS, in which the local search phase is performed by a VND that uses different neighbourhood structures to the ones used for the shake phase. Skewed VNS can be considered a minor variant of VNS. In this case, the solution obtain from the local optimisation process is accepted to reallocate the search taking into account not only its fitness but also its distance respect to the current solution.

2.2.6 Evolutionary Computation

Evolutionary Computation (EC) is based on natural selection and genetic principles and was proposed by Holland [78]. These algorithms evolve a

Algorithm 7 Evolutionary Computation.

procedure *EvolutionaryComputation*()
  Generate initial population P
  Evaluate all individuals in P
  while not stopping condition do
    $P\prime \leftarrow$ **Crossover**(P)
    $P\prime\prime \leftarrow$ **Mutation**(P')
    Evaluate all individuals in $P\prime\prime$
    $P \leftarrow$ **Selection**($P\prime\prime \cup P$)
  end while
  return best individual found

population of solutions (also called individuals) applying certain operators to create the population of the new generation. The operators are:

- *Crossover*: it combines parts of two or more parent solutions in order to create new solutions that possibly can improve their parents.
- *Mutation*: it modifies randomly a solution to increase the diversity and generate better individuals.

Apart from crossover and mutation operators another important process in EC is the selection. Those individuals that show a better adaptation have a higher probability of being chosen to continue in the next generation or of being combined to create the individuals of the next population.

Multiple EC algorithms have been proposed from their origins until now. Some of the most know are Evolutionary Programming [56,57], Evolutionary Strategies [128] and Genetic Algorithms [66, 78, 112, 129, 148]. The majority of the current variants of Evolutionary Programming are used for continuous optimisation, as most of the Evolutionary Strategies. On the contrary, Genetic Algorithms are usually applied to combinatorial problems.

An example of a basic EC method is displayed in Algorithm 7. The working usually carried out by evolutionary methods is, in first place, to create a new population of individual P. Then, at each generation, a new population is obtaining by the recombination, mutation and selection of individuals.

The main components of a EC method are the following [17]:

- *Individuals of the population*: EC algorithms work with populations

of individuals. These individuals are usually solutions for the problem at hand, but other options are possible: partial solutions, set of solutions or objects that give rise to solutions. The binary representation of solutions is the most common one, although representations by permutations or real numbers are also possible.

- *Evolutionary process*: this process determines which individuals will belong to the next generation. The selection scheme can vary from the generational replacement, where only the offspring of the former generation is chosen, to the selection of distinct proportions of individuals of the previous population and the offspring. Keep a elitist set of solutions is a common characteristic of EC method with a fixed number of individuals.

- *Neighbourhood structure*: which solutions can be combine among them is another aspect to define in EC. When any individuals can be chosen to apply the crossover operator, then the population is non structured. There exists other models, as the cellular genetic algorithms [5], where the population is structured and the individuals can only be combined with those ones that belong to their neighbourhood.

- *Information sources*: although the most common crossover operators uses two solutions to generate new offspring, others operators that combine more than two parents have been proposed [49]. In works as [115, 142], we can find mechanisms that generate new individuals using population statistics.

- *Infeasibility*: in some cases the recombination of individuals can lead to non feasible solutions, that is, solutions that do not fulfill the problem constraints. In this way, it is necessary to define mechanisms that manage these situations. The most simple option is to reject these individuals, although in problems where the generation of feasible solutions is hard (scheduling problems), a strategy that keeps these individuals but penalizes them as a function of their quality, usually leads to better results. Another possibility is to repair non feasible solutions [50].

- *Intensification strategy*: EC methods are good global searches but present some problems to intensify the search in promising regions.

This has lead to many researchers to incorporate local search in EC algorithms. The most known combination between EC and local searches are the so called memetic algorithms [75, 114]. The local search help to quickly detect favorable areas of the search space. Another way of improving the intensification capabilities of EC methods is to use crossover operators that combine promising parts of the parents instead of establish randomly which parts are taken to create the offspring. This strategies are named linkage or building block learning [74, 151].

- *Diversification strategy*: a drawback that EC algorithm can show is the premature convergence to suboptimal solutions, specially if they are coupled with local searches. Some techniques have been proposed to overcome this problem. The mutation operator is the most basic element to control the diversity of a population, so a relatively easy way of increase/decrease the diversity is to adjust the probability of application of these operator [7]. Other strategies designed to deal with this drawback are crowding and preselection [98]. Fitness sharing is another mechanism to enhance the diversification skills of EC methods. This consists on reducing the fitness of individuals that are densely concentrated in a region of the search space [67].

2.2.7 Ant Colony Optimisation (ACO)

These optimisation techniques were firstly proposed by Dorigo et al. [43, 44, 46] and are inspired in the behaviour of some social insects. These insects were taken as source of inspiration since despite their individual simplicity, they present a quite organized social structure that allow them to achieve tasks that one isolated individual would not be able to do by itself. Ant Colony Optimisation (ACO) studies models to simulate in somehow the behaviour of real ants by means of populations of artificial agents in order to solve optimisation and distributed control problems.

Different aspects of ants' social behaviours (search of food, work division or cooperative transport), that have inspired some of these algorithms, can be explained by the stigmergy, concept introduce by Pierre-Paul Grassé [70]. This author defined the term as "Stimulation of workers by the performance reached". In short, this concept indicates that the action done by some agents left signals in the environment that are perceived by other agents

Algorithm 8 Ant Colony Optimisation (ACO).

procedure *AntColonyOptimisation*()
  Initialize pheromone
  while not stopping condition do
    Build solutions of ants
    Update pheromone
  end while

and that can determine future actions. Some studies have shown that many self-organizational behaviour of social insects can be explained only by the stigmergy. One of the most known examples the search and transport food done by ants. When they look for a feeding source, they left a chemical substance (pheromone) in the ground that is perceived by the others ants. These insects tend to follow those paths with a bigger concentration of pheromones. Such behaviour leads them to transport food to their niche in very effective and efficient way, since they find quasi optimal paths.

Last phenomenon is simulated by ACO algorithms to solve optimisation problems. For example, many of these methods are based on the models developed to explain the behaviour of ants when they need to choose between two different bridges (with the same or different length) that finish in the same feeding source [42, 69, 120]. A mathematical model to estimate the probability that an ant takes the first bridge (p_1) or the second one (p_2), assuming that m_1 and m_2 ants have pass through the bridges number 1 and 2, respectively, was proposed by Goss [69]. This is given by the next equation:

$$p_1 = \frac{(m_1 + k)^h}{(m_1 + k)^h + (m_2 + k)^h} , \tag{2.1}$$

where the parameters k and h are fixed according to the experimental data. The probability p_2 is calculated by $p_2 = 1 - p_1$.

ACO, whose basic scheme can be seen in Algorithm 8, is a constructive metaheuristic where the solutions are built by adding components to partial solutions following a probability distribution that is determine by the pheromone and heuristics of the particular problem. The working of these methods is the following: in first place the pheromone and different parameters are initialised. After this, ACO goes into a two-phase loop. In the first of these two phases, a set of m ants constructs solutions by adding

elements from a finite set of solution components C. Starting from a empty solution, at each iteration the solution is extended by joining a component that does not violated the constraints. This subset of C is named $N(s)$. The construction process can be modeled as a path in a graph $G_C(V, E)$ where the elements of the set C are represented as vertices or edges. The election of a component from $N(s)$ is done using a stochastic mechanism that has as one of its parameters the pheromone associated to each element of $N(s)$. Different stochastic models to accomplish this election has been developed, all of them taking as base the one seen in Equation 2.1.

In some ACO algorithms, after constructing all the solutions, a local search process is applied to improve the performance of the method. The use or not of local search usually depends on the problem and the ACO algorithm.

The last phase in an ACO method is the update of the pheromone. The objective of this phase is to increase the levels pheromone for promising solutions and decreasing (evaporate pheromone) for the ones that are not.

The first ACO algorithm that can be found in the literature is the so called Ant System (AS) [45] in which all ants participate in the pheromone update. MAX-MIN Ant System is a variant of this method where some elements are improved. For example, the pheromone traces are only updated by the best ant and furthermore, the pheromone values are delimited empirically and adapted to problem specific features. Another algorithm developed from AS ideas is the Ant Colony System. In this case, the ants perform a local update of the pheromones independent from the one done during the construction process. After each step, the ants do this local update by decreasing the pheromone quantity of the last edge visited in order to increase the probability that other ants take different paths and so, improving the diversification of the search.

2.2.8 Particle Swarm Optimisation

Particle Swarm Optimisation (PSO) is a metaheuristic inspired on the social behaviour showed by some species of birds or fishes when they accomplish some tasks as looking for food. This method, firstly developed by Eberhart and kennedy in 1995 [87], maintains a set of solutions, called particles, that are points within the search space of the problem. The particles, after being initialized randomly, search for a local optimum guided by the particle that

Algorithm 9 Particle Swarm Optimisation (PSO).

```
procedure ParticleSwarmOptimisation()
  Initialize all particles in population P by setting their velocity
  and location
  while not stopping condition do
    for all particle ∈ P do
      Evaluate(particle)
    end for
    for all particle ∈ P do
      UpdateBestPosition(particle)
    end for
    UpdateBestGlobalPosition()
    for all particle ∈ P do
      UpdateVelocity(particle)
      CalculateNewPosition(particle)
    end for
  end while
```

has found the best solution (the leader) and by the best solution found during their trajectories.

The basic form of this metaheurstic can be seen in Algorithm 9. This strategy, after the initialisation stage, enters in a loop that finishes when the stopping condition is met. As first step of this loop, the particles are evaluated and then, the best solution found by each of them ($pbest_i$) and the best global position ($gbest$) are updated. Afterwards, the velocity of each particle is recalculated by means of the next equation:

$$v_i = \omega \cdot v_{id} + c_1 \cdot r_1 \cdot (pbest_i - x_i) + c_2 \cdot r_2 \cdot (gbest - x_i) \tag{2.2}$$

where ω is a parameter called inertia weight that establishes the influence of the previous velocity. A higher value implies a bigger diversification balance of the strategy. The constants c_1 and c_2 are named acceleration coefficients although they are also known as the cognitive and the social parameters, respectively, since they determine the importance of the best local and global positions, in that order. r_1 and r_2 are random values uniformly distributed in the range [0,1]. Finally, in the last step of the loop the new position of the particle is calculated by:

$$x_i = x_i + v_i \tag{2.3}$$

PSO is a metaheuristic that besides having a very simple tuning (due to its low number of parameters) shows a high performance, which facilitates its application to different fields as neural network training [47], fuzzy control systems [47], dynamic problem optimisation [15], etc.

Different versions of the PSO algorithm have been proposed. One example is multi-swarm PSO [15], where the particles are divided in different subsets that compete among them. In charged PSO [14], some of the particles have assigned a "electrostatic charge" whereas others are neutral. As it happens in an atom, particles equally charged show a repulsive force among them while on the contrary neutral particles do no experience this force. The aim behind this idea is to gather neutral particles around the best global position at the same time as charged particles explore the areas close to this position.

2.2.9 Parallel Metaheuristics

In some occasions the computational times associated to the resolution of large instances can be extremely high. In this way, the application of parallel computation to Metaheuristics appeared in a natural way in its development not only to speed up the resolution of big instances but also to build methods more robust that offer an acceptable performance in a wide number of problem or instances [35].

In this context, Parallel Metaheuristics can be classified in 3 categories according to the source of parallelism used:

- *Type 1 or low level parallelism*: this source of parallelism is commonly found within an iteration of the search method. It can be obtained through the concurrent execution of the operations or the concurrent evaluation of several movements that are accomplished during the iteration of the search algorithm. The target of this strategy is to reduce the computational time and not to improve the quality of the solution or changing the search pattern of the method with the intention of achieving a better exploration of the search space. Furthermore, the reader should note that given the same quantity of iterations, both the sequential and the parallel implementations produce the same results. Some of this implementations modify the sequential method to exploit the extra computation power available, but without altering the basic algorithm. One common example is to evaluate at the same

time all the elements of the current solution's neighbourhood instead of just one. This type of parallelisation has been applied to methods as genetic algorithms [55] or simulated annealing [18] among others.

- *Type 2 or domain decomposition*: in this category, the parallelism is obtained decomposing the decision variables into disjoint subsets. Each process works on one of these subsets and sets the others to a fix value. It is important to note that this way of partitioning the decision variables may not allow to explore the whole search space, which obliges in some occasions to repeat the division of the variables and the search process. This strategy is generally implemented in a master-slave model, where the master accomplishes the task of partitioning the variables. The division is adjusted in every restart or at specific intervals. Depending on the chosen method, the slaves can explore concurrently their assigned subset keeping fixed the other variables or can have access to the whole set. In the last case, the master should accomplish complex tasks to combine the partial solutions obtained in each subset and this way, build a complete solution of the problem. Tabu Search and GRASP have been parallelize following this methodology in [38] and [51], respectively.

- *Type 3 or multi-search*: In this parallelization strategy several searches explore the solution space concurrently. Each search thread can implement the same or a different method, can start from the same or a different initial solutions, etc. If the threads only exchange information at the end of the search to determine the best solution, then the strategy is called independent. If, on the contrary, the search algorithms send and receive information during the search process then the strategy is called cooperative multi-search. They can work with different communication modes: synchronous, asynchronous, event-driven, predetermined or dynamically chosen moments, etc. The speed up in computation time is achieved by reducing the time of search for the threads in a factor proportional to the number of available processors. Furthermore, this type of parallelization is often used to increase the exploration capabilities of a metaheuristic. Examples of these approaches can be seen in [33, 36].

2.3 Summary

In this chapter we have presented the Soft Computing area, reviewing its classic components. We have also seen that Metaheuristics should be considered as one of its main elements since, in this manner, the research field is extended allowing a higher number of methods and models in the part of Functional Approximation and Optimisation Methodologies than when only Evolutionary Algorithms are considered.

In the second part of the chapter we have focused on the component of Soft Computing that frames our dissertation, Metaheursitics. Here, we have defined some basic concepts of optimisation. Afterwards, we have described some of the most known Metaheuristics as Simmulated Annealing, Tabu Search, Greedy Randomized Adaptive Search Procedure, Variable Neighbourhood Search, Evolutionary Computation, Ant Colony Optimisation, Particle Swarm Optimisation and Parallel Metaheuristics.

Chapter 3

Cooperative Strategies for Optimisation

In this chapter we will start describing Hybrid Metaheuristics, different types of hybridization according to a well known taxonomy as well as a review of some of approaches that we can find within each of these types. Taking as general context Hybrid Metaheuristics, we will define what will be our concept of cooperative strategy locating these methods within that general context. Three different classifications for cooperative strategies will be given and we will finish with a summary of the content of the chapter.

3.1 Hybrid Metaheuristics

The first stages of research in metaheuristics were focused in the design and application of "pure" or isolated metaheuristics. The researchers considered the metaheuristics that they were working with as "generally" best, and they tend to follow specific philosophies strictly. However in last years an increasing interest in the combination among different metaheuristics or among these methods and others optimisation techniques has arisen. This new type of search algorithms, called *hybrid metaheuristics*, can provide more robust performance than their "pure" counterparts when dealing with real-world and large-scale problems. In fact, many of the best results reported in the literature for practical or academical problems have been obtained by these class of metaheuristics. A proof of the success of these new methods can be seen in Figure 3.1 where we show of the number of publica-

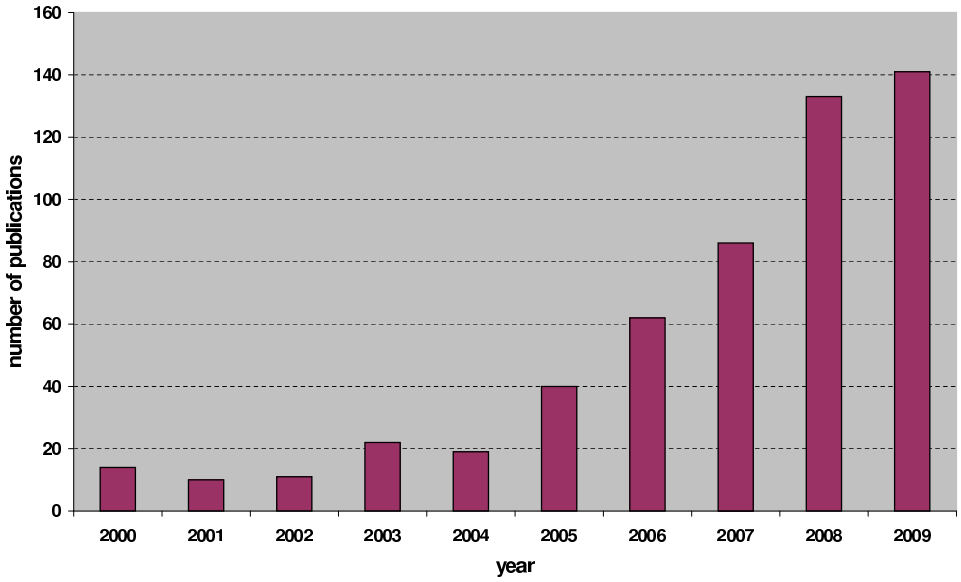


Figure 3.1: Evolution during last ten years of the number of publications (journal and conference papers) within the field of Computer Science that contain in the title the word hybrid plus one of the following terms: heuristic, optimization, metaheuristic and algorithm

tions within the field of Computer Science that contain in the title the word hybrid plus one of the following terms: heuristic, optimization, metaheuristic and algorithm [1]. The graphic clearly show that the number of journal and conference articles about this subject has grown noticeably in the last five years.

The motivation behind the hybridization of different algorithms is the development of better performance strategies by combining the advantages of "pure" or isolated methods. In the literature we can find combinations of many different forms which have lead several authors to "order" them attending to different criteria.

From our point of view, one of the most interesting taxonomies was given by E.G Talbi in [143]. It is a hierarchical taxonomy that can be seen in Figure 3.2, where the hybrids are classified by their structure. At the first level, we can differentiate between Low-level and High-level hybridizations. Low-level combinations refer to those cases where a determined component

[1]The data have been obtained from SCOPUS: http://www.scopus.com

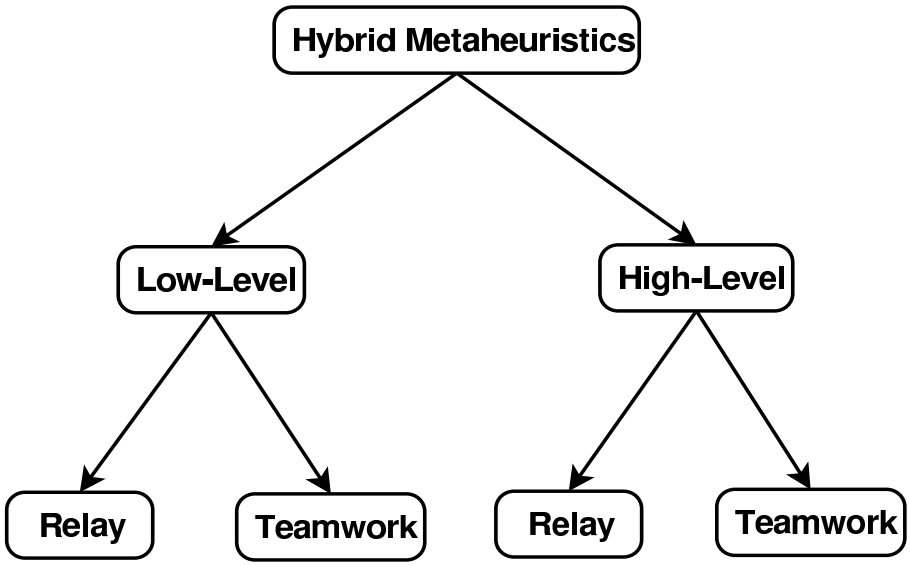


Figure 3.2: Hybrid metaheuristics taxonomy

of a metaheuristic is substituted for another metaheuristic, which leads to a strong dependency between them. On the contrary, in High-level hybrids, the algorithms keep their original identity and are able to work in an autonomous way. Within the next level, relay hybridization includes those methods where algorithms are applied sequentially in a pipeline, that is , the output of one algorithm is connected to the input of other one. On the other hand, teamwork hybrids represent cooperative optimisation models where a set of agents cooperates while each of them accomplishes its own search. In this way we can differentiate 4 different classes of hybrid metaheuristics according to their structure. Next subsections are devoted to describe in more detail these classes and some examples.

3.1.1 Low-Level Relay Hybrid (LHR)

This class includes those methods in which a a non-general purpose mechanism is embedded into a metaheuristic, usually to carry out the local improvements. Examples of these methods are hardly found in the literature. Two samples of this hybrid can be seen in [2], where a memetic algorithm is provided with a 2-opt optimisation, and [103], where a simulated annealing algorithm is coupled with a deterministic local search that carries out the exploration of the local optima.

3.1.2 Low-level Teamwork Hybrid (LTH)

LTH hybridization usually combines metaheuristics that have an exploratory profile, as population-based algorithms, with trajectory based heuristics, that have a higher exploitation capability, due to the fact that they have complementary strengths and weaknesses. We can find many proposals where the most successful population-based algorithms have integrated, into their working scheme, trajectory-based metaheuristics such as tabu search, simulated annealing or hill climbing. The idea in these cases is to perform a more global search by means of the first type of methods while the last ones carry out a more local optimisation. These algorithms have shown a very good results in many optimisation problems and in fact, most of the high-performance population-based metaheuristics are coupled with local search methods.

A common example of these hybrids is an evolutionary algorithm that substitute their classic blind operators by heuristics that use the individuals as starting points to later replace them by the obtained solutions that usually have a better quality. For example, the mutation operator of a genetic algorithm has been substituted by a trajectory search as Hill Climbing [86], Tabu Search [144] or Simulated Annealing [28]. In [29] we can find a local search which incorporates problem specific knowledge and that is used instead of the genetic operators. Within this type of operators we should differentiate between Lamarckian, where the individual is replaced by the local minimum found, and Baldwinian, where the local optimum is simply considered to evaluate the individual. LTH hybrids have improved other methods in hard problems as Graph Coloring [54] where a genetic algorithm combined with a local search provided good results.

Classic crossover operators that do not use heuristic information have been also replaced, for instance, by methods that incorporate problem specific information [71] or a greedy algorithm [97]. Similar types of crossovers can be found in continuous optimisation. In [154] the authors introduce a heuristic crossover that determines the direction of the search basing on the fitness of the parents. Another example is presented in [130] where the Simplex algorithm is used to combine solutions.

Besides evolutionary algorithms, this hybridization can be done with other population-based methods as ant colonies [141], genetic programming [118], particle swarm optimisation, etc. In all of these cases, the trajectory

algorithms aims to intensify the search.

These hybrids can show premature convergence problems in some occasions, especially if the trajectory-based method is applied in every iteration. Conditional hybridization can avoid this stagnation by using the local optimizer when a preestablished criterium is fulfilled. Such conditions can be static, if the hybridization is applied with a determine frequency, or dynamic, if the application is addressed by events (e.g no improvement for a given number of iterations).

3.1.3 High-Level Relay Hybrid (HRH)

This class includes those hybrid metaheuristics composed by a set of self-contained algorithms that are run sequentially and whose output is connected to the input of the next one. One basic example, which gives usually very good results, is the generation of the starting solution of a trajectory methods by means of a greedy heuristic. A similar scheme can be applied to population metaheuristics to generate the initial population and, in this way, achieving a high diversity as happens in scatter search [64].

Trajectory-based and population-based metaheuristics can also be combined as HRH hybrids. Population algorithms usually perform a global search which makes them locate promising regions quickly. But once they have located the region, its capability to intensify the search in those areas is low. Therefore it would be useful to apply a local search in that locations. Furthermore, a usual situation in population based method is to have a quite uniform population after a certain amount of time which leads to a low probability of finding more fitted individuals. This is generally due to the stagnation of the population in a basin of attraction. The fast exploitation of these basins is advisable and thus, the use of trajectory algorithms since it is experimentally proved that they can perform better that task. In this sense, the local optimizers can be applied to:

- *The whole population*: It has a big computational cost but leads to the best final solution.
- *A subpopulation*: Here, a compromise between the search time and the quality of the best solution found is established. The selection of the population can be done following criteria as the diversity of the population, best individuals, etc.

- *The best solution of the population:* The trajectory metaheuristic is applied to the best solution of the population. In this case the computational complexity is reduced but it is not ensured to reach the best final solution.

Other methods of intensification as path relinking [4] can be used in this type of hybridization. This can also be applied to the whole population or to a subset of elite solutions.

In the literature we can find examples of these methods, as [93] where a Genetic Algorithm is introduced to improve the solution obtained by Simulated Annealing. In [99] a Simulated Annealing is used to increase the quality of the solutions found by a Genetic Algorithm. Other example are the so-called multi-memetic algorithms [75], where a set of methods (operators, local searches, metaheuristics) are coupled with a Genetic Algorithm. Hyperheuristics [27] also fit this type of hybridization. These methods consist on a set of low-level heuristics managed by a specific heuristic or plan that determines which one should be applied in every moment. This decision is made using information about the performance of the low-level heuristics.

3.1.4 High-Level Teamwork Hybrid (HTH)

In HTH a set of autonomous cooperating agents perform a parallel search to obtain good quality solutions. In some occasions the agents only exchange information at the end of the search to determine the best solution, which is usually named as "independent strategy". In many other cases the agents cooperate by sending and receiving information during the search process. They can work with different communication modes: synchronous, asynchronous, event-driven, predetermined or dynamically chosen moments, etc. Moreover, this type of hybrids increase the exploration capabilities of a metaheuristic. Some works as [33,36] showed that HTH hybridization allows to obtain a more effective behaviour (better quality solutions) than their isolated components. In these studies, we can also check that the combination of agents implementing different search algorithms leads these strategies to a more robust performance in relation to the variation of the instance problem characteristics. They also indicate that while the implementation of the independent search is easier and their results are good, these can be improved by introducing a cooperation scheme.

Within this hybrid class, the well-known island models of Genetic Algorithms [25] are also included, where a set of sub-populations evolve exchanging individuals among them . Several parameters control these HTH hybrids:

- *Topology*: how the different populations are connected
- *Migration rate*: number of individuals that migrate
- *Replacement strategy*: which individuals are replaced by the solutions sent from other subpopulations
- *Migration interval*: frequency of the migrations

In the majority of the models the solutions sent to other subpopulations are copies of the original individuals but some authors has proposed models in which the migrants really "leave the island".

An island model for permutation problems is presented in [111] with the peculiarity that the individuals are initialized by a heuristic in order to generate well distributed and fully disjoint initial population in each island. Also, Ref. [156] describes another island model for the shortest path routing problem.

HTH hybrids can be also applied to trajectory based metaheuristics. A common model for this hybridization is to run in parallel several local optimizers and establish a cooperation scheme. For instance, Pelta et al. proposed in [125] an hybrid where a set of trajectory algorithms are controlled by a rule-driven coordinator. A similar strategy where a set of tabu searches cooperate can be found in [85]. Here the cooperation is accomplished by maintaining a global reference set which uses the information exchange to promote both intensification and strategic diversification.

The so called "Algorithm Portfolios" [82] is other type of HTH hybrid. Here, a set of algorithms are run in parallel until the fastest one solves the problem. Algorithms Portfolios are often coupled with other techniques from Artificial Intelligence in order to control the amount of resources available for each component which is usually done taking into account its performance. Such techniques can be reinforcement learning [90], dynamic programming [126] or bandit problem solvers [59] among others.

We can also find HTH hybrids of others metaheuristics as Variable Neighbourhood Search [33], ant colonies [100], differential evolution [91], etc.

3.2 Locating cooperative strategies within hybrid metaheuristics context

The concept of cooperative strategy is widely used within the field of metaheuristics, being taken by many authors to name different types of methods. Crainic and Toulouse define them as [37]:

"*a set of highly autonomous programs (APs), each implementing a particular solution method, and a cooperation scheme combining these APs into a single problem-solving strategy*"

The next definition is given by Blum et al. [17]:

"*Cooperative search consists of a search performed by agents that exchange information about states, models, entire sub-problems, solutions or other search space characteristics*"

In other works the authors consider cooperative strategies as a type of parallel metaheuristics:

"*These algorithms execute in parallel several search programs on the same optimization problem instance*" [146]

or

"*Cooperative search algorithms are parallel search methods that combine several individual programs in a single search system*" [145]

The concept of cooperative strategy that we will use along the thesis will be the same that Talbi call "teamwork" in his taxonomy, that is: *models in which we have many parallel cooperating agents, where each agent carries out a search in a solution space.* Therefore, the term "cooperative strategy" in this dissertation embraces two classes of that taxonomy: LTH and HTH. Another point that we should highlight is the fact that the particular implementation of the cooperative strategy is not taken into account, that is, we include both parallel and sequential implementations of these methods.

Next section is devoted to explain different ways of classifying these type of strategies.

3.3 Classifying cooperative strategies

The majority of the taxonomies given to classify cooperative strategies are not exclusively dedicated to this type of methods but consider them as one of their classes. For instance, in [143], apart from the hierarchical classification we saw before, Talbi gives a flat taxonomy for hybrid metaheuristic that can be applied to cooperative strategies. He consider the following categories:

- *Homogeneous vs Heterogeneous*: this class refers to the composition of the strategy. When all the combined methods use the same metaheuristic the hybrid is homogeneous. Island models for Genetic Algorithms [25] are good examples of this hybridization. In these cases, different parameters are usually used for the methods. An example can be found in [149] where a set of cooperating tabu searches start with distinct settings (initial solution, tabu list size, etc.).

  On the contrary, heterogeneous strategies obviously use different algorithms. Combinations between evolutionary methods and local searches [34, 75] are probably the most known instances of this class.

- *Global vs Partial*: this is another perspective from which cooperative strategies can be classified. Global refers to those methods where the algorithms explore the whole solution space, being the most common case in the literature.

  Partial strategies decompose the search space of the problem and each of the algorithms that constitute the strategy is assigned a specific sub-space. This usually implies the existence of a mechanism to keep the feasibility of the solutions (constraints, domain ranges, etc). An example of this class of methods is shown in [83], where a set of Genetic Algorithms are applied to job-shop scheduling problem. Here, each algorithm evolves individuals of a specie which represent the process plan for one job.

- *Specialist vs General*: in general hybrids, all the components work with the same objective function which is the common case in the literature.

On the contrary, specialist methods are formed by algorithms that solve different problems. This hybridization can be found in [6]. This approach for the quadratic assignment problem is composed by Tabu Searches that work over this problem whereas a Genetic Algorithm carry out a diversification task which is formulated as another problem. A different case of specialist hybrids, relatively common, is the use of a metaheuristic to optimise the parameters of another one. This approach has been applied to optimise Genetic Algorithms [136] or ant colonies [3] by Genetic Algorithms.

Another taxonomy for hybrid metaheurisitcs was proposed by G. Raidl in [127] where he unified other classifications previously given for this type of algorithms. As the former one, it can also be applied to cooperative strategies. It considers four different classification criteria:

- *What is hybridized?*: from this point of view three distinct types of cooperative strategies can be considered: combination of metaheuristics with metaheuristics, metaheuristics with problem specific algorithms or metaheuristics with other techniques coming from Operational Research or Soft Computing methods.

- *Level of hybridization*: in these cases, two types of cooperative strategies can be differentiated. On one hand, high-level or weak coupling when the algorithms maintains their identities and on the other hand, low-level or strong coupling, when individual components are exchanged.

- *Order of execution*: by this criterion we have *batch* model, where the algorithms are run sequentially and the information is always exchanged in one direction; *interleaved*, where the components are executed also sequentially but the interactions can be done in different directions; and *parallel*, that besides from running the algorithms simultaneously can show many forms of communications among them.

- *Control strategy*: two different control strategies can be found. The first one embraces integrative approaches in which one algorithm act as an embedded component of another. The second one is collaborative o cooperative control where none of the algorithms is a component of the others.

To finish with this section we are going to give another interesting way of classifying cooperative strategies that was proposed in [35]. This taxonomy for parallel metaheuristics, which generalise the ones presented in [32, 39], defines three dimensions: Search Control Cardinality, Search Control and Communications and Search Differentiation. Last two are related to the cooperation issue. The first dimension distinguishes the algorithms in which the global search is controlled by an individual process (*1-C*) from those where is done by several processes that work in a collegial manner (*p-C*). Since cooperative search strategies belong to this last category, from now on, we will not consider the *1-C* class.

The next dimension, *Search Control and Communications*, addresses the classification in 4 classes taking into account the quality and the quantity of information exchange. These are:

- *Rigid synchronization (RS)*: it indicates that there is little or no information exchange between processes. The independent strategy would be framed here.

- *Knowledge synchronization (KS)*: in this degree of the Search Control and Communication dimension is characterised by synchronous operating mode. In this case the self-contained search algorithms independently explores the search space and stop at a previously determined intervals where an intensive communication phase takes place among all of them.

- *Collegial(C)*: in this stage, the search threads perform an asynchronous communication. Each search algorithm is able to process, store and treat its own information and is provided with mechanisms to decide when to accomplish the information exchange. Here, each solver searches on all or on a part of the domain and when an improving solution is found (locally or globally, according to the chosen strategy), it is broadcasted (together, eventually, with its own context and history)to all or some of the other search processes. This information can be also stored in a central memory and it is only broadcasted that a better solution has been found. In all cases, the messages sent correspond to the messages received.

- *Knowledge Collegial (KC)*: as in the former case, the searches have its own capabilities to deal with the information and to determine when

to communicate. However, in this stage the exchanged information is analysed and processed in order to infer new knowledge about the global search process. To accomplish this inference, the majority of these methods implement some form of global memory (pool, blackboard or data warehouse) through which the information exchange is done.

The third dimension, called *Search Differentiation*, is related to the number of different starting solutions/populations and to the number of different strategies implemented by the solvers (by different strategies this classification means distinct algorithms or same algorithm with distinct configurations). The authors identify the next four classes:

- *Single (Initial) Point/Population Single Strategy (SPSS)*: it is the most simple case, and it generally allows for only low level parallelism.
- *Single Point/Population Different Strategies (SPDS)*: in this case there are heterogeneous search threads that start from the same solution/s.
- *Multiple Points/Populations Single Strategy (MPSS)*: it refers to those strategies where the solvers start the exploration from different points but implementing the same algorithm.
- *Multiple Points/Populations Different Strategies (MPDS)*: this class is the most general and includes all others as specific cases.

3.4 Summary

In this chapter we have introduced cooperative strategies. To that end, we have started describing the general context in which these strategies are framed, Hybrid Metaheuristics. In order to have a comprehensive vision of this context we have used a taxonomy to explain the different types of hybridization taking into account their structure. This classification was organized in a hierarchical way where in the first level distinguished between high-level and low-level hybrids. In the second level there are another two classes, relay and teamwork. In this way we have four different categories for each one of which some approaches proposed in the literature have been reviewed.

Besides this, three different taxonomies of cooperative strategies has been given. Although they were proposed to classify a more general set of methods, they can also be applied to this sort of metaheuristics. In first place we gave a flat taxonomy presented by Talbi where the algorithms are classified attending to criteria as the composition, search space assign to each agent and diversity among the objective functions the agents work with. Next classification was proposed by Raidl and takes into account four aspects of the cooperative strategies: what is hybridized, level of hybridization, order of execution and control strategy. Finally, we commented a taxonomy given by Crainic et al. where two out of three dimensions were devoted to the cooperation issue: Search Control and Communications, and Search Differentiation.

Taking into account the general context of Hybrid Metaheuristics, we explained what will be our concept of cooperative strategy along this dissertation. It is given by the notion of teamwork in Talbi's taxonomy, that is, *models in which we have many parallel cooperating agents, where each agent carries out a search in a solution space.* In this way we define the frame in which the cooperative strategies that will be presented in the next chapters are located.

Part III

Thesis Contributions

Chapter 4

Centralised cooperative strategies: the roles of solvers definition and cooperation scheme

This chapter presents a study of two key aspects of centralised cooperative strategies, the definition of the cooperating solvers and the cooperation scheme established among them. The first issue, to be more concrete, is focused on analysing the behaviour of the strategy when uses the same optimisation algorithm or different ones. As for the next aspect, a new cooperation scheme is proposed and compared against one previously presented. The results obtained using the Uncapacitated Single Allocation p-Hub Median Problem as test bed give some insights about the performance of homogeneous vs heterogeneous cooperative strategies and about the influence of the cooperation scheme as well as showing the benefits of the new scheme of cooperation.

4.1 Motivations

Within cooperative strategy paradigm, we can find methods with many different forms but all of them share two essential characteristics: a set of search algorithms to deal with the problem and a cooperation scheme to create a synergy among them. We can say that these two elements compose the core

of a cooperative strategy and therefore, they determine its performance in an important way. With this idea in mind, in this chapter we aim to make an in-depth study of those two components, in the context of combinatorial optimisation problems, in order to have a better understanding about them. In this sense, we will continue with the research done a centralised cooperative strategy based on Soft Computing that was previously developed in the MOdels of Decision and Optimization research group (MODO). This strategy is composed by a set of solvers that are controlled by a rule-driven coordinator. Apart from analysing new aspects, the research done here is tested on a new problem, the Uncapacitated Single Allocation p-Hub Median Problem (USApHMP), and the solvers coordinated by the strategy implements different algorithms.

As we saw in 3.3, the set of cooperating agents(also called solvers) can be homogenous or heterogeneous if all of them implements the same resolution algorithm or a different one, respectively. How these two type of composition affect to the performance of the cooperative strategy will be the first point to study.

The second issue tackled in this chapter will be the cooperation scheme. Our analysis will focus on the control rule base of the coordinator since it is the main element of the cooperation scheme for the strategy used here. To this end, a new control rule based on Reactive Search [8] is presented. Reactive Search is a framework where optimization techniques are coupled with a machine learning component that analyses the behaviour of these algorithms and provides feedback by fine tuning its parameters. To asses the performance of the new scheme, this is compared against a fuzzy rule previously presented in [40, 125]. The comparison also allow us to analyses the behaviour change experienced by the strategy when different cooperations schemes are used.

The chapter is structured as follows. In Section 4.2 the centralised cooperative strategy used for this study is described, as well as the two different control rules considered. Section 4.3 is devoted to the experimental framework. Concretely, the USApHMP problem is defined and the implementation details of the heuristics used as solvers and of the strategy are given. As for the analysis of the results, we will study in first place the composition of the strategy in Section 4.4 and subsequently, the cooperation scheme in 4.5. Finally, the conclusions are discussed in 4.6.

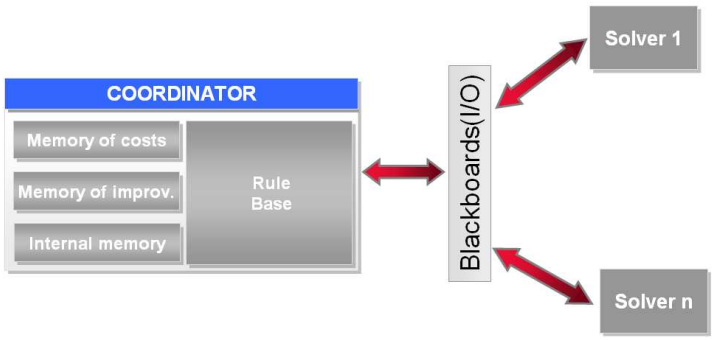


Figure 4.1: Scheme of the cooperative strategy

4.2 Description of the centralised cooperative strategy

The proposed studies will be conducted over a centralised cooperative strategy [40, 125] whose general scheme is presented in Figure 4.1. There is a set of solvers, where each solver can implement the same or a different resolution strategy for the problem at hand. The coordinator processes the information received from the solvers and produces subsequent adjustments of their behaviour by sending "orders". To achieve this exchange of data, a blackboard model [52] is used. Concretely, two blackboards are available, one where the solvers write the reports of their performance and the coordinator reads them, and another, where the orders are written by the coordinator and read by the solvers.

After an initialization stage, the solvers execute asynchronously while sending and receiving information. The coordinator checks through the input blackboard which solver provided new information and decides whether its behaviour needs to be adapted using a rule base. If this is the case, it will calculate a new behaviour which will be sent to the solvers through the output blackboard. As for the solvers, its working it is also very simple: once execution has begun, performance information is sent and adaptation orders from the coordinator are received alternatively.

Every cooperative strategy of this kind should define an information management strategy providing: the type of information that solvers send, the type of information the coordinator sends, and how such information is processed. Such details are described in the next subsection.

4.2.1 The information management strategy

As we have just seen, the information flow in our strategy can be divided in the three following steps: 1) performance information is sent to the coordinator from the solvers, 2) this information is processed and stored by the coordinator and 3) coordinator sends directives to the solvers. In this subsection we are going to describe which information is managed in every step and which transformations are made.

In the data flow solvers $\Rightarrow$ coordinator, each report contains the next items:

- solver identification;
- a time stamp t;
- the current solution of the solver at that time s^t;
- the best solution reached until that time by this solver s_{best};
- a list with the local minima found by the method since the last report.

The coordinator stores the last two reports from each solver. From these two reports, the coordinator calculates its improvement rate as:

$$\Delta_f = \frac{f(s^t) - f(s^{t'})}{t - t'} \tag{4.1}$$

where $t - t'$ represents the elapsed time between two consecutive reports, $s^{t'}$ is the current solution sent by the solver in the last report and f is the objective function. The values Δ_f and $f(s^t)$ are then stored in two fixed length ordered data structures, one for the improvements and the other for the costs. The list of local minima is processed by the coordinator that keeps the history of all local optima in a hash table. Each entry of this table has also a collision counter with the number of times that a solution has been visited by any search thread.

The behaviour of the solvers is controlled by a set of rules. These rules allow the coordinator to determine if a solver is behaving correctly, as well as the action that should be performed to correct such behavior. These rules are of the type:

if condition **then** action.

As we will deal with trajectory-based solvers, the *action* part of the rule takes the form

action: send a new solution to the $solver_i$

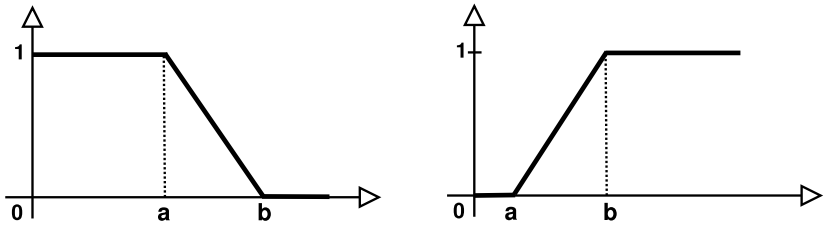
When the current solution of a local search method is changed, the search trajectory continues from a new point in the search space. This is the behaviour implied by the *action* defined here. As *new solution*, several alternatives are possible: sending a new one, for example, a randomly generated new one, or the best one seen by the coordinator up to time t. As the strategy progresses, the solvers will therefore located in those regions of the solution space determined by the coordinator in order to intensify or diversify the search depending on the cooperation scheme or on the state of the solvers.

4.2.2 Cooperation schemes

The cooperation scheme of this strategy is mainly defined by the control rule that the coordinator uses. In this subsection we will describe two control rules that will be used in the next part of these chapter. The first one is a fuzzy rule that has been previously developed within the group MODO being used in works as [40, 125] where yielded good results on different problems. The second one is presented here and was designed basing on *reactive search* [8] ideas.

The term *reactive search* refers to an algorithmic framework where optimization techniques are coupled with machine learning algorithms. In particular, the machine learning component analyzes the behavior of the optimization algorithm and provides feedback by fine tuning its parameters, thus adapting it to the properties of the instance being solved. Parameter tuning can be performed:

Offline: the machine learning component analyzes the behavior of the optimization algorithm after a series of runs on different instances. The purpose of this method is to learn a mapping between some instance features and a satisfactory value of the algorithm's parameters. In this case, the algorithm simply replaces the researcher in performing offline adjustments when he applies an algorithm to a new domain, with a trial-and-error approach.



a) Definition for improvement rates

b) Definition for solution costs

Figure 4.2: Membership function of the two sets used in the fuzzy control rule.

Online: the machine learning component operates alongside with the optimization algorithm and tries to detect hints of bad performance, such as repeated visits to the same configuration or low improvement rate. By performing online adjustments to the optimization component, the system can adapt to the local features of the search landscape.

Fuzzy rule

This rule was designed following the principle: "If a solver is working well, keep it; but if a solver seems to be trapped, do something to alter its behaviour". Its precise definition is as follows:

> **if** the quality of the current solution reported by $solver_i$ is *low* and the improvement rate of $solver_i$ is *low* **then** send perturbed C_{best} to $solver_i$

The labels *low* are defined as fuzzy sets whose membership functions $\mu(x)$, for both improvement rates and solution costs, are shown in Figure 4.2. The variable x corresponds to the relative position (resembling the notion of percentile rank) of a value (an improvement rate or a cost) in the samples stored in the respective fixed length data structure. The parameters (a, b) are fixed to $(80, 100)$ and $(0, 20)$ for the data structure of costs and improvements, respectively. This way of measure the quality of the improvement rates and the solution is independent of the problem, instance or scale. C_{best} denotes the best solution ever recorded by the coordinator.

What the rule says, is that, if the values reported by a solver are among the worst stored in the two sets, then the coordinator sends the solution C_{best} (with a small perturbation) to the solver. By doing so, it relocates

the solver to a more promising region of the search space, trying to increase the chances of finding better solutions. The perturbation applied to C_{best} is usually done by a mutation operator.

Reactive Control Rule

Following the principles in [10, 11], this rule uses the history of visited local minima to detect search stagnation. In other words, the hash table kept by the coordinator is used to determine if a solver is visiting an already explored area of the search space. If this is the case, the coordinator drives the searchers diversification by restarting the stagnated algorithm from a perturbation of the best configuration found across all solvers. The definition of this rule is the following:

> **if** the collision counter cc of the last local minima visited by $solver_i$ is bigger than $\lambda_{reaction}$, **then** the coordinator sends C_{best} to $solver_i$ perturbed by degree ϕ.

The threshold $\lambda_{reaction}$ regulates the activation of the rule and ϕ is defined as:

$$\phi = \begin{cases} cc - \lambda_{reaction}, & \text{if } cc - \lambda_{reaction} < \phi_{max} \\ \phi_{max}, & \text{if } cc - \lambda_{reaction} \geq \phi_{max} \end{cases}$$

The idea of this rule is that the more often a local minimum is visited, the higher the probability that it belongs to a large attraction basin, and therefore the perturbation needs to be higher in order to escape from that optimum. The strength of the perturbation is controlled by the application of different mutation operators.

4.2.3 Classifying the strategy

To finish the description of the centralised cooperative strategy, we are going to classify it according to the three taxonomies given in Section 3.3:

- **Talbi's taxonomy**:
  - *hierarchical taxonomy*: high-level teamwork hybrid
  - *flat taxonomy*: general and global method that can have both homogeneous and heterogeneous nature.
- **Raidl's taxonomy**:

  - *what is hybridized?*: the algorithm presents two hybridization types in this sense. On one hand, metaheuristics are combine with metaheuristics in the set of solvers, and on the other hand, this set of solvers is hybridized with fuzzy logic or reactive search depending on the cooperation scheme.
  - *level of hybridization*: high-level
  - *order of execution*: interleaved
  - *control strategy*: collaborative

- **Crainic et al's taxonomy**:
  - *search control cardinality*: p-C
  - *search control and communications*: KC since the information is exchanged through the coordinator which uses this information to infer when a solver is showing a bad performance.
  - *search differentiation*: MPSS or MPDS depending on the definition of the solvers.

4.3 Experimental Framework

This part of the chapter is devoted to show different aspect of the experimentation carried out to achieve the objectives proposed. We will start defining the test bed used, the Uncapacitated Single Allocation p-Hub Median Problem (USApHMP). Next, the search methods used as solvers will be described and finally, further implementation details will be given.

4.3.1 The Uncapacitated Single Allocation p-Hub Median Problem

Hub location problems appear when is necessary to guide the flow from an origin to a destiny, but it is not recommendable or very expensive to have a connections between each pair origin-destiny. The objective of this kind of problems is composed by two steps:

- **Hub location:** to determine which nodes should be the hubs and the number of them, in order to distribute the flow across them.

- **Non-hub to hub allocations:** to assign the rest of the nodes to the hubs.

Generally, these task are performed by minimising an objective function that describes the exchange flow and its cost. A general survey of this kind of problems can be found in [24].

Here we focus on the Uncapacitated Single Allocation p-Hub Median Problem (USApHMP). Uncapacitated means the hubs have not constraint on the amount of traffic, single indicates the nodes can only be assigned to a unique hub; and p-Hub signifies the number of hubs is fixed to p.

In this chapter we are going to use the quadratic integer formulation given by O'Kelly in [116]. Let N be a set of n nodes. We define W_{ij} as the amount of flow from the node i to j, and C_{ij} the cost of transporting a unit between the nodes i and j.

Let X_{ij} be decision variable defined as

$$X_{ij} = \begin{cases} 1 & \text{if node } i \text{ is allocated to hub } j \\ 0, & \text{otherwise} \end{cases}$$

The USApHMP can be formulated as:

$$min \sum_{i,j,k,l \in N} W_{ij}(\chi C_{ik} X_{ik} + \alpha C_{kl} X_{ik} X_{jl} + \delta C_{jl} X_{jl}) \tag{4.2}$$

subject to

$$\sum_{j=1}^{n} X_{jj} = p \tag{4.3}$$

$$\sum_{j=1}^{n} X_{ij} = 1, \forall i = 1, \ldots, n \tag{4.4}$$

$$X_{ij} \leq X_{jj}, \forall i, j = 1, \ldots, n \tag{4.5}$$

$$X_{ij} \in \{0, 1\}, \forall i, j = 1, \ldots, n \tag{4.6}$$

Parameters χ, α and δ stand for the cost of collection (generally $\chi = 1$), the cost of distribution (generally $\alpha < 1$) and the cost of transfer (generally $\delta = 1$). The objective function (4.2) minimises the sum of the origin-hub,

hub-hub and hub-destination flow costs. Constraint (4.3) ensures that exists exactly p hubs. (4.4) indicates that a node can only be allocated to a unique hub. Condition (4.5) guarantees that a non-hub point can only be allocated to a hub and not to another non-hub node. Finally, (4.6) is the classical binary constraint.

The USApMP belong to the class of the NP-hard problems. Moreover, although the set of hubs is fixed, the assignment sub-problem is also NP-Hard [96].

The instances chosen for the experimentation were obtained from the resource ORLIB [12]. Concretely, we used the AP (Australian Post) data set derived from a study of the postal delivery system. The instances utilised can be divided in two groups, those with 50 o less nodes, and those with more than 50. For the first set were considered instances with 2, 3, 4 and 5 hubs, while for the second one the different numbers of hubs were 2, 3, 4, 5,10,15 and 20. The value of the constants χ, α and δ are fixed to 3, 0.75 and 2 respectively. The optimum for those instances with a number of nodes less than 50 was provided by the resource ORLIB, and for the other instances we considered the best solution found by one of the state-of-art algorithms for this problem presented in [89].

4.3.2 Description of the solvers

In this subsection, we will show the implementation of the different heuristics used as optimisation algorithms in the experiments that will be seen later on. Firstly, the neighborhood operator will be described as it forms the core of the three different heuristic searches which were chosen: Tabu Search, Simulated Annealing (SA) and Variable Neighborhood Descent search(VND).

The cited heuristics used their very basic definitions. Moreover, all of them, were implemented by the author as a lab assignment for a metaheuristics course in the Computer Science Engineering degree of the University of Granada. Before starting with the descriptions, we are going to show some aspects of the used notation:

- $f(s)$ is the fitness value of the solution s.
- $G_j = \{i | X_{ij} = 1, i \neq j\}$. G_j is the group of those nodes that are allocated to hub j.
- s_{curr} is the current solution of the heuristic.

- s_{best} represents the best solution obtained by the algorithm until that moment.
- $\mathcal{N}(s, O)$ symbolises a neighbourhood of the solution s, that is, a set of solutions generated from s by means of the operator O. This operator can be seen as a function $O : S \times P \rightarrow S$, being S the solution space and P a set of tuples of parameters.
- checkBestSol(s, s_{best}) is a procedure which updates s_{best} to s if the value of the objective function for s is lower than for s_{best}. Otherwise, this procedure does nothing.

Neighbourhood operator

The neighbourhood operator is composed of two distinct mechanisms: assignment change of non-hubs nodes and location change of hubs. The first of them, consists on changing the allocation of a non-hub node to a different hub. The following steps are the next:

1. Choose randomly a group G_j
2. Select randomly a node $i \in G_j$
3. Choose randomly another group G_k, $k \neq j$
4. Allocate the selected node to the new group: $X_{ij} \leftarrow 0, X_{ik} \leftarrow 1$

The other mechanism changes the location of a hub j to other node that is currently allocated to such hub. If there is no nodes allocated to j, other node is selected as hub and j is assigned to other group. To do this change the next stages are followed:

1. Choose randomly a group G_j
2. If there is at least one node in the group ($|G_j| > 0$) then:
   (a) Select randomly a node $i \in G_j$
   (b) Allocate all nodes in G_j and its hub node j to the new hub node i: $\forall k \in G_j$: $X_{kj} \leftarrow 0$, $X_{jj} \leftarrow 0$, $X_{ki} \leftarrow 1$, $X_{ji} \leftarrow 1$ and $X_{ii} \leftarrow 1$
3. If the group has no nodes ($|G_i| = 0$), then:

Algorithm 10 Variable Neighbourhood Search solver pseudocode

procedure $VariableNeighbourhoodSearchSolver$

$s_{curr} \leftarrow$ **GenerateInitialSolution()**
$best \leftarrow s_{curr}$
$k \leftarrow 1, iter \leftarrow 1$
repeat
  Generate a neighbourhood $\mathcal{N}(s_{curr}, O_{p_k})$ **of** N_{VND} **solutions**
  $neigh_{best} \leftarrow$ **ChooseBest(**$\mathcal{N}(s_{curr}, O_{p_k})$**)**
  if $f(neigh_{best}) < f(s_{curr})$ **then**
    $s_{curr} \leftarrow neigh_{best}$
    checkBestSol(s_{curr}**,**$best$**)**
    $k \leftarrow 1$
  else
    $k \leftarrow k + 1$
  end if
  if $k > k_{max}$ **then**
    $k \leftarrow 1$
  end if
  $iter \leftarrow iter + 1$
until $iter > iter_{max}$
return $best$

(a) Choose randomly another group $G_k, k \neq j$ with at least one node.

(b) Select randomly a node $i \in G_k$.

(c) Make a new group with the selected node i. $X_{ii} \leftarrow 1$

(d) Allocate the last hub j as a normal node to another hub selected randomly. $X_{jr} \leftarrow 1$ where r is a random hub.

From now on, we will refer to this operator as O_p, where $p = (a, b) \in P$ is a pair of values, being a and b the number of assignment changes of non-hub nodes and location changes of hub nodes, respectively.

Implementation of VND search

We are going to begin with the description of the VND heuristic. The pseudocode of this search is given in Algorithm 10:

Algorithm 11 Simulated Annealing solver pseudocode

procedure *SimulatedAnnealingSolver*

$s_{curr} \leftarrow$ **GenerateInitialSolution()**
$best \leftarrow s_{curr}, iter \leftarrow 1$
$T_0 = \frac{\mu}{(-log(\sigma))} * f(s_{curr}), temperature = T_0, t \leftarrow 0$
repeat
  $s_{curr} \leftarrow$**ExploreNeighbourhood(**s_{curr}**,** $temperature$**)**
  checkBestSol(s_{curr}**,**$best$**)**
  $iter \leftarrow iter + 1, t \leftarrow t + 1$
  $temperature \leftarrow \frac{T_0}{1+t}$
until $iter \geq iter_{max}$
return $best$

Three different pairs of values p_k for the neighbourhood operator were considered. These were $p_1 = (1, 1)$, $p_2 = (2, 1)$ and $p_3 = (3, 1)$. As it can be seen in the pseudocode, the search finishes after $iter_{max}$ iterations.

Implementation of SA

The SA heuristic implemented it is a very simple version which uses the annealing schema of Cauchy. Its pseudocode is given in Algorithm 11.

As former heuristic, the search is over when reaches $iter_{max}$ iterations. The function *explore_neighbourhood*, that can be seen in Algorithm 12, takes over the exploration of the neighbour solutions with a determined temperature:

The function *metrop* is defined as:

$$metrop(\delta, t) = \begin{cases} true, & \text{if } \delta < 0 \text{ OR } u() < e^{\delta/t} \\ false, & \text{otherwise} \end{cases} \tag{4.7}$$

where $u()$ is a random number uniformly distributed in the interval $[0, 1]$. As for the operator O_p used we should say that p is equal to $(1, 1)$ and its working has some peculiarities. This only makes one of the two type of changes each time, concretely, makes a change of assignation with probability 0.4 or of localisation with probability 0.6.

Algorithm 12 ExploreNeighbourhood pseudocode

procedure *ExploreNeighbourhood*

$s'_{best} \leftarrow s_{curr}$
for $i \leftarrow 0$ **to** $annealings_{max}$ **do**
Generate a neighbourhood $\mathcal{N}(s_{curr}, O_p)$ **with one solution.**
Select the solution s_{cand} **from the former neighbourhod.**
$\delta \leftarrow f(s_{cand}) - f(s_{curr})$
if $metrop(\delta, temperature)$ **then**
$s_{curr} \leftarrow s_{cand}$
end if
end for
return s'_{best}

Implementation of Tabu Search

To finish with this subsection, tabu search will be described. The pseudocode is displayed in Algoritm 13

The operator O_p used is the same as the former heuristic. When the best non-tabu neighbourhood is chosen, its re-assigned node is stored in TL_{assig} or its re-localised hub is put in TL_{local}, depending on the type of change done.

The generation of the neighbourhood is achieved using the two tabu lists TL_{assig} and TL_{local}. These lists are used to mark as tabu those neighbours whose reassigned nodes are in TL_{assig} or whose re-localised hubs are in TL_{local}. We should add that if a tabu neighbourh fulfils the aspiration level, then its tabu label is removed. This aspiration level is satisfied when the objective value of such solution is lower than $f(best)$. As same as the two last algorithms, it finishes when the search has done $iter_{max}$ iterations.

4.3.3 Further details of the cooperative strategy

When implementing this type of cooperative strategies, one can resort to real parallel implementations, or simulate the parallelism in a single-processor computer. The later is the strategy adopted here: we construct an array of solvers and we run them using a round-robin scheme. In this way, each of them is run for a certain number of evaluations of the objective function.

Algorithm 13 Tabu Search solver pseudocode

procedure *TabuSearchSolver*

$s \leftarrow$ **GenerateInitialSolution()**
$best \leftarrow s$
$iter \leftarrow 0$
Initialise the tabu lists TL_{assig} **and** TL_{local} **with the sizes** $size_{assig}$ **and** $size_{local}$**, respectively.**
repeat
PermittedSet(s) $\leftarrow \{s' \in \mathcal{N}(s, O_{tabu})$ | **tabuConditions**(s')=$false$ **OR aspirationCriteria**(s')=$true$**}**, $|\mathcal{N}(s, O_{tabu})| = N_{tabu}$
$s \leftarrow$ **ChooseBest(PermittedSet**(s)**)**
CheckBestSol(s, $best$)
until $iter \geq iter_{max}$
return $best$

This number is randomly generated from an interval *freq_interval*. Once a solver is executed, the communication with the coordinator takes place. These steps are repeated until the stop condition is fulfilled.

Regarding the fuzzy rule base, firstly, the size of the memory of costs and improvements was set to 4 per the number of solvers, that is, both memories have a capacity of nine elements. It is triggered when the output value of the antecedent is higher than 0.9. The modification done to C_{best}, when it is sent to the solvers, corresponds with one assignment change and one location change. For the Reactive rule, $\lambda_{reaction}$ was set to 10. The parameter ϕ_{max} is set to 3, that is, 3 different degrees of perturbation for C_{best} are considered. This degree of perturbation is determined by the number of changes of assignment and location applied to this solution. When the degree $\phi = i$, then the operator applied to the solution is $O_{(k,k)}$ being $k = number\ of\ hubs/(\phi_{max} - i - 1)$, $i = 1, \ldots, \phi_{max}$.

As for the setup of the different heuristics, the values assigned to the different parameters can be seen in Table 4.1.

Next part is devoted to the analysis of the results. As we stated in the motivation of the chapter, this analysis is two-fold. Firstly, the importance of the solvers definition will be studied and secondly, the influence of the cooperation scheme.

Metaheuristic	parameter = value
VND	$k_{max} = 3$ $N_{VND} = 80$
SA	$\mu = 0.4$ $\sigma = 0.4$ $annealings_{max} = 20$
Tabu	$size_{assig} = n/8$ $size_{local} = n/8$ (n is the number of nodes)

Table 4.1: Parameters' settings for VND, SA and Tabu Search

4.4 Analysing the influence of the solvers definition

To accomplish this analysis the next two types of cooperative strategies (homogeneous vs heterogeneous) and isolated solvers were compared:

1. *SA*: isolated simulated annealing strategy
2. *Tabu*: isolated tabu search strategy
3. *VND*: isolated variable neighborhood descent search strategy
4. *H-SA*: cooperative strategy composed by 3 identical copies of SA
5. *H-Tabu*: cooperative strategy composed by 3 identical copies of Tabu
6. *H-VND*: cooperative strategy composed by 3 identical copies of VND
7. *Heterogeneous*: cooperative strategy composed by one SA, one Tabu and one VND

Every strategy is run 30 times (each one starting from a different initial solution) and each run finishes when 25000 evaluations for instances with 50 nodes or less, and 200000 for instances with more than 50 nodes were used. Instance with 10 nodes or 2 hubs are omitted due to its extreme ease that allows to obtain the optimum in all the run with all the methods. In this case the interval that determines the frequency of communication (*freq_interval*)was set to [100, 150]. At the end of each run, we measure an error as $error = 100 \times \frac{obtained\ value - optimum}{optimum}$.

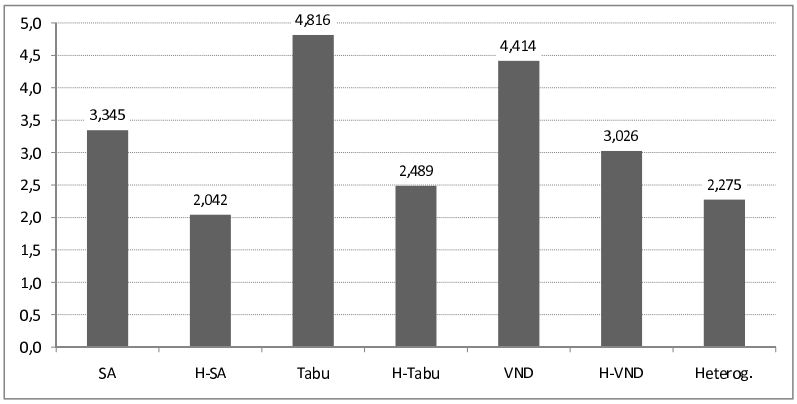


Figure 4.3: Average error over 24 instances obtained by the seven methods compared.

A global view of the results can be seen in Figure 4.3, where for every method, the average error obtained over all the instances is displayed. Viewing this graphic, a substantial improvement can be observed when cooperation is used. Looking at the isolated vs homogeneous cooperation strategies, we can see that *H-Tabu* experienced the largest decrease in error (48%), following by *H-SA* (38%) and *H-VND* (31%). The Mann-Whitney's U non-parametric test with $\alpha < 0.05$ confirmed that these differences have statistical significance. If we compare the heterogeneous method versus the homogeneous strategies, we can say that the average global error obtained by the first one is significantly better than the achieved one by *H-VND*. However, when such comparison is done versus either *H-SA* or *H-Tabu* instead of *H-VND*, the differences are not significant (same non-parametric test).

A detailed analysis per instance is shown in Tables 4.2 and 4.3 where the average error for both the individual methods and their homogeneous cooperative versions are displayed. The corresponding improvement percentage is also shown.

Nds	Hbs	SA	H-SA	imp SA	Tabu	H-Tabu	imp Tabu
20	3	2,89(2,15)	0,10(0,33)	97*	2,24(2,20)	0,72(1,41)	68
	4	3,42(1,98)	2,01(1,54)	41*	1,41(1,85)	1,45(1,55)	-3
	5	1,82(1,66)	0,74(1,13)	59*	0,25(0,67)	0,50(0,79)	-99
25	3	2,17(2,26)	0,79(1,27)	64*	2,42(2,01)	1,12(1,65)	54*
	4	1,84(2,45)	0,43(0,83)	77	2,16(2,36)	0,82(1,72)	62*
	5	4,48(3,52)	1,44(2,72)	68*	2,56(3,55)	1,01(2,35)	60*
40	3	0,43(0,91)	0,32(0,72)	26*	0,83(1,27)	0,32(0,72)	62
	4	1,60(2,54)	0,00(0,00)	100	2,35(2,81)	0,30(1,16)	87
	5	2,37(1,72)	1,18(1,58)	50*	3,72(2,73)	2,05(1,08)	45*
50	3	1,99(2,34)	0,51(1,03)	75*	3,73(2,38)	1,04(1,76)	72*
	4	2,28(3,06)	0,69(2,11)	70	6,17(3,23)	1,54(2,66)	75*
	5	4,87(2,63)	2,90(2,06)	40*	6,93(3,41)	4,33(1,47)	37*
100	3	0,52(1,06)	0,61(1,12)	-16*	2,78(1,80)	0,87(1,25)	69*
	4	1,96(3,02)	0,00(0,00)	100*	4,45(2,42)	0,22(1,14)	95*
	5	2,55(1,25)	2,13(0,60)	16	3,84(2,03)	2,19(0,42)	43*
	10	5,56(3,10)	2,88(2,04)	48*	9,15(3,63)	4,13(2,39)	55*
	15	7,96(2,17)	5,95(2,32)	25*	11,29(2,79)	6,70(2,43)	41*
	20	6,24(2,38)	5,10(2,12)	18	8,41(2,37)	4,78(1,89)	43*
200	3	0,23(0,70)	0,35(0,78)	-52*	0,79(0,95)	0,00(0,00)	100*
	4	1,83(3,12)	0,23(1,13)	87	1,60(2,47)	0,44(1,57)	72*
	5	1,24(0,46)	1,10(0,59)	11*	2,21(2,39)	1,02(0,34)	54*
	10	4,56(2,36)	4,04(1,93)	11	8,34(3,50)	4,12(1,88)	51*
	15	8,33(2,42)	8,07(1,81)	3	12,58(2,40)	9,93(2,55)	21*
	20	9,13(2,87)	7,43(2,06)	19*	15,38(3,19)	10,13(2,31)	34*

Table 4.2: Average(std. deviation) for individual and homogeneous cooperative searches, and percentage of improvement (imp) for each instance ($(avg.invidual - avg.cooperative)/avg.individual * 100$). (*) indicates that the improvement is significant (Mann-Whitney's U non-parametric test $\alpha < 0,05$)

Nds	Hbs	VND	H-VND	imp VND
20	3	1,16(2,15)	0,19(1,05)	83*
	4	2,54(1,35)	1,43(1,31)	44*
	5	2,43(2,47)	0,70(1,00)	71*
25	3	1,92(1,74)	0,11(0,55)	94*
	4	3,73(3,60)	1,87(2,45)	50
	5	7,40(4,13)	4,43(3,45)	40*
40	3	0,57(1,11)	0,00(0,00)	100*
	4	3,54(3,09)	0,79(1,83)	78
	5	3,52(2,71)	2,60(1,60)	26
50	3	3,13(2,32)	0,79(1,57)	75*
	4	4,46(3,98)	3,65(3,04)	18
	5	5,87(3,35)	4,28(1,31)	27*
100	3	1,02(2,05)	0,09(0,47)	92*
	4	1,68(2,82)	0,37(1,39)	78
	5	2,46(0,80)	2,13(0,45)	13
	10	9,42(3,91)	5,14(2,69)	45*
	15	9,51(2,80)	7,57(2,54)	20*
	20	7,85(3,21)	7,00(2,46)	11
200	3	0,66(1,28)	0,17(0,61)	74*
	4	1,74(2,87)	0,39(1,20)	78
	5	1,68(1,40)	0,98(0,28)	42*
	10	7,02(2,37)	5,56(2,05)	21*
	15	10,76(2,64)	10,64(1,69)	1
	20	11,87(3,23)	11,75(2,19)	1

Table 4.3: Average(std. deviation) for individual and homogeneous cooperative searches, and percentage of improvement (imp) for each instance $((avg.invidual - avg.cooperative)/avg.individual * 100)$. (*) indicates that the improvement is significant (Mann-Whitney's U non-parametric test $\alpha < 0,05$)

Focusing on the simulated annealing algorithm, we can observe that the homogeneous cooperation leads to better results in almost all cases. For those instances with less than 50 nodes, we can find decreases in average error above 25 percent, reaching the 100 percent in 40n-4h (40 nodes and 4 hubs). Only in two cases (25n-4h and 50n-4h) the differences in the average error are not significant. When the size of the instances is enlarged, the degree of improvement is lower although it is still important. This is higher than 10 percent except for 200n-5h where the enhancement falls to 3%. From a statistical point of view, in 5 out of 12 cases (100n-{5,10}h; 200-{4,5,15}h) the cooperation does improve significantly the individual SA. It is also interesting to note that there exist two cases 100n-3h and 200n-3h, where cooperation produced worst results. The reason is as follows: as the same cooling scheme is used both in the individual as in the cooperative scheme and in the last one, each solver has 1/3 of the time available, the cooperative strategy can not effectively reach a search phase where intensification is performed. The cooperative search concentrates the trajectory around the good points, but when sampling neighbor solutions (due to the not so low temperature) transitions to worse solutions can be easily accepted.

When the cooperation is done with a set of tabu search solvers, the improvement with respect to an isolated tabu search is notorious. For those instances with 50 or less nodes, the improvement produced by the cooperation is greater than 37% except for 20n-{4,5}h where the individual metaheuristic obtains a better performance. In statistical terms, all the differences between homogeneous cooperation and isolated algorithms are significant with the exception of instances with 20 nodes and 40n-3h. If we take as reference the other group of instances, the ratio of improvement varies from 21% to 200n-15h and 100% in 200n-3h, being all of them statistically significant.

When VND is analyzed, it is easily seen that cooperation produced equal or better results than the individual method. In small instances the percentage of improvement goes from 26% in 40n-5h to 100% in 40n-3h. If we consider instances with 100 or more nodes, we found that in the two biggest ones (200n-{15-20}h) the improvement obtained by cooperation is just slightly appreciable, only a 1%, and obviously is not statistically significant. However, in the remaining cases, the enhancement obtained by homogeneous VND is superior to 10%, even reaching a 92% in 100n-3h. Apart of the two biggest instances, just seven cases are not statistically significant. These are

Nodes	Hubs	heterogeneous
20	3	0,12(0,68)*
	4	1,47(1,51)
	5	0,25(0,51)
25	3	0,42(1,04)*
	4	0,24(0,17)*
	5	1,87(2,49)
40	3	0,13(0,48)
	4	0,50(1,53)
	5	1,65(1,25)
50	3	1,02(1,60)
	4	1,00(2,17)
	5	3,96(1,77)

Nodes	Hubs	heterogeneous
100	3	0,26(0,79)
	4	0,22(1,14)
	5	2,07(0,60)*
	10	3,56(1,90)*
	15	6,60(2,30)
	20	4,91(1,63)*
200	3	0,07(0,38)
	4	0,70(1,90)*
	5	0,97(0,49)
	10	3,99(1,89)
	15	9,25(1,80)
	20	9,35(2,45)

Table 4.4: Average(std. deviation) for the heterogeneous cooperative search in each instance. (*) indicates that heterogeneous search is significantly better than the best individual metaheuristic (Mann-Whitney's U non-parametric test $\alpha < 0,05$).

25n-4h, 40n-{4,5}h, 50n-4h, 100n-{4,20}h and 200n-4h.

In short, these results say that using just three copies of an individual method plus a cooperation scheme, one can obtain an optimization scheme that almost always will lead to an important improvement in performance.

When the cooperative strategy is made up from different solvers, the analysis should be slightly changed. The average error and the standard deviation obtained by the heterogeneous cooperative strategy are shown in Table 4.4. We marked those instances where the method obtained better results than the best individual method (the minimum of each row among $SA, Tabu, VND$ in Tables 4.2 and 4.3). Again, cooperation allowed to obtain equal or better results than the individual methods.

To conclude the analysis, we show in Figures 4.4, 4.5, 4.6, 4.7, 4.8 and 4.9 a number of scatter plots providing pairwise comparison among the seven methods evaluated. Every instance is plotted as a pair (x, y) where x (y) is the average normalized error obtained by the strategy named in the X (Y)

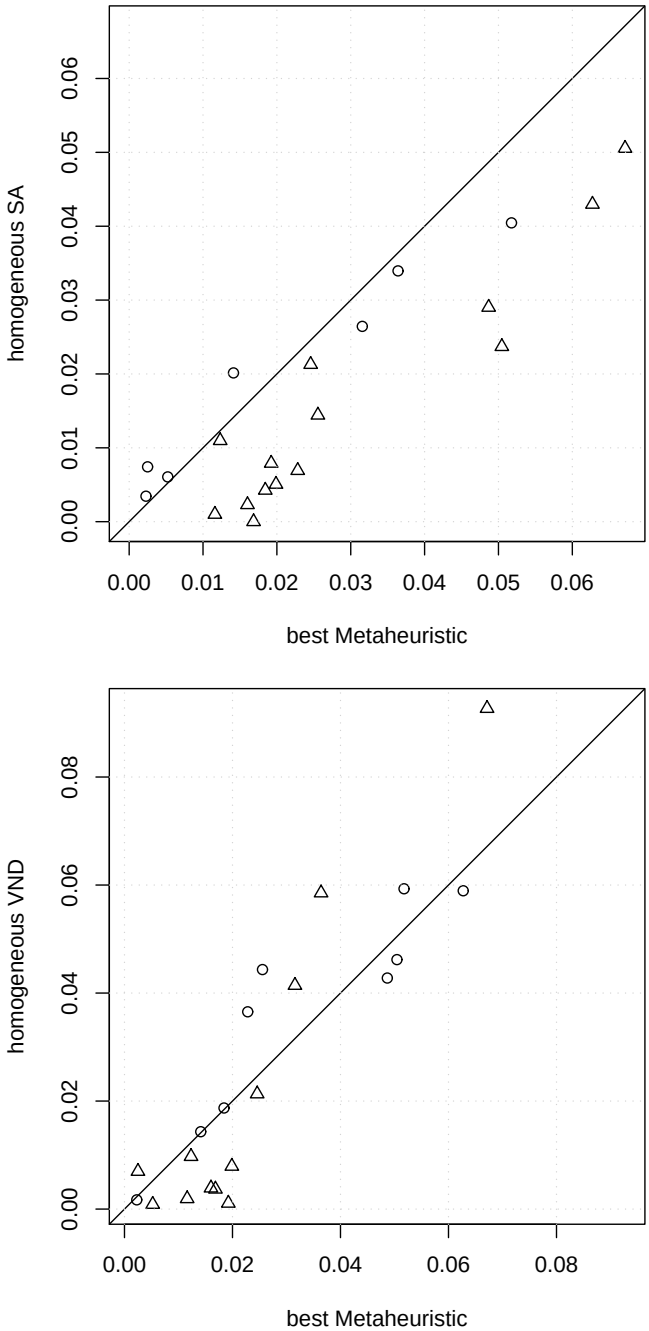


Figure 4.4: Cooperative Strategies vs. Best Metaheuristic: comparison of the average deviation from the optimum (values are normalized). A triangle represents an instance where both algorithms perform differently at significance level 0.05 (Mann-Whitney U test).

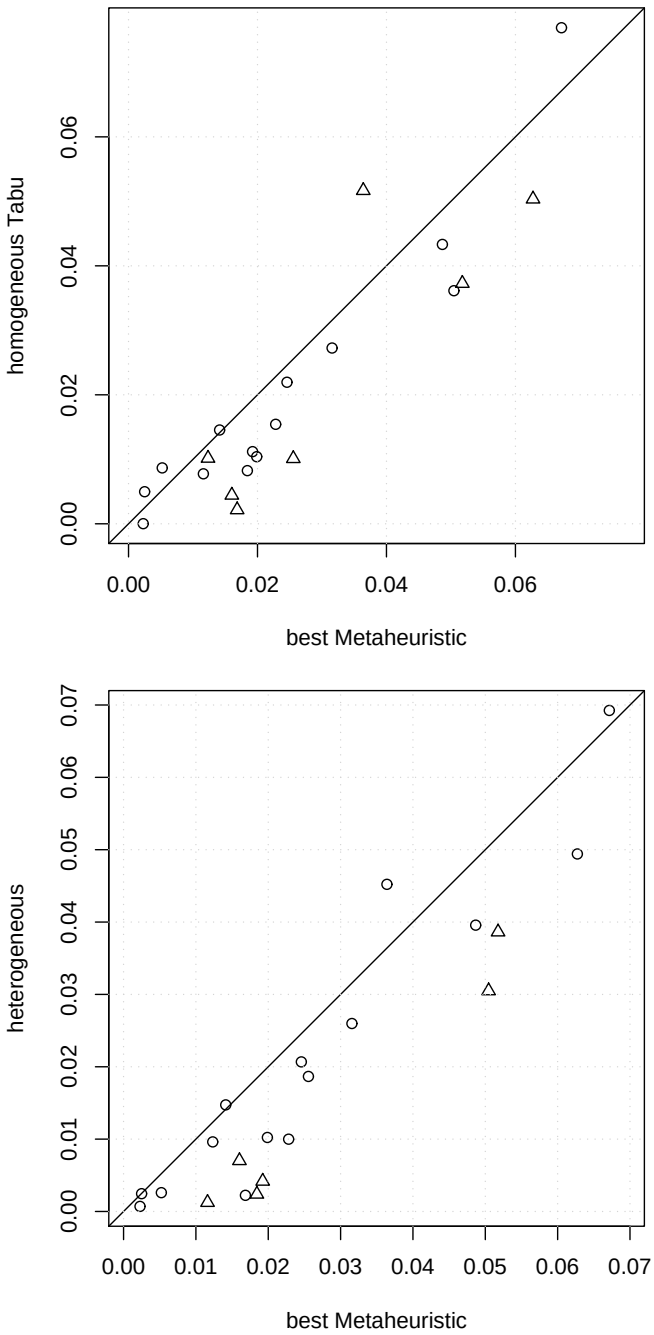


Figure 4.5: Cooperative Strategies vs. Best Metaheuristic: comparison of the average deviation from the optimum (values are normalized). A triangle represents an instance where both algorithms perform differently at significance level 0.05 (Mann-Whitney U test).

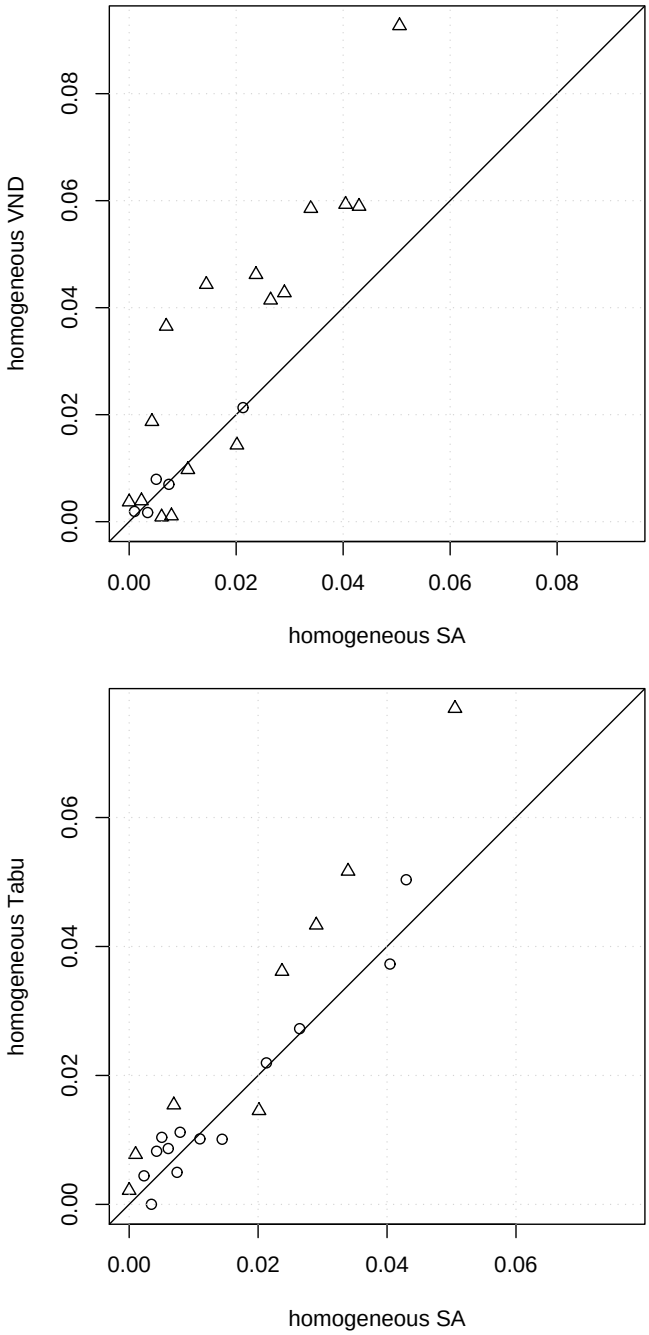


Figure 4.6: *H-Sa* vs. *H-Tabu* and *H-Sa* vs *H-VND*:comparison of the average deviation from the optimum (values are normalized). A triangle represents an instance where both algorithms perform differently at significance level 0.05 (Mann-Whitney U test).

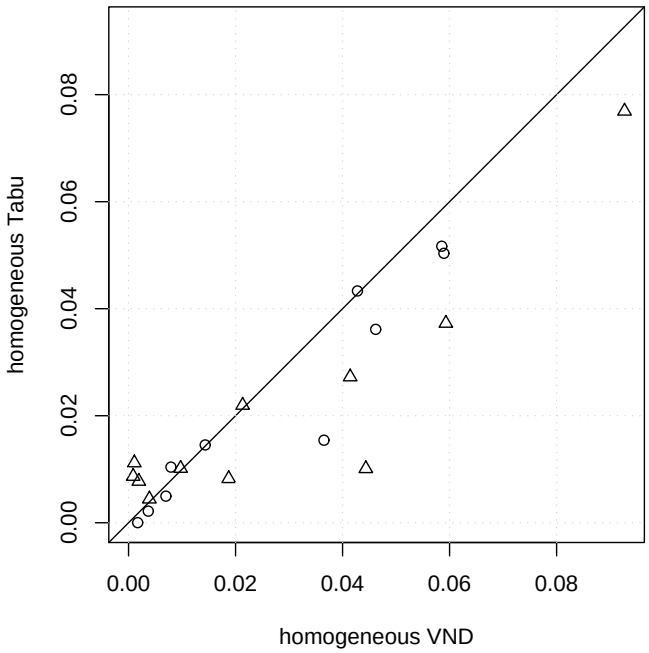


Figure 4.7: *H-Tabu* vs *H-VND*:comparison of the average deviation from the optimum (values are normalized). A triangle represents an instance where both algorithms perform differently at significance level 0.05 (Mann-Whitney U test).

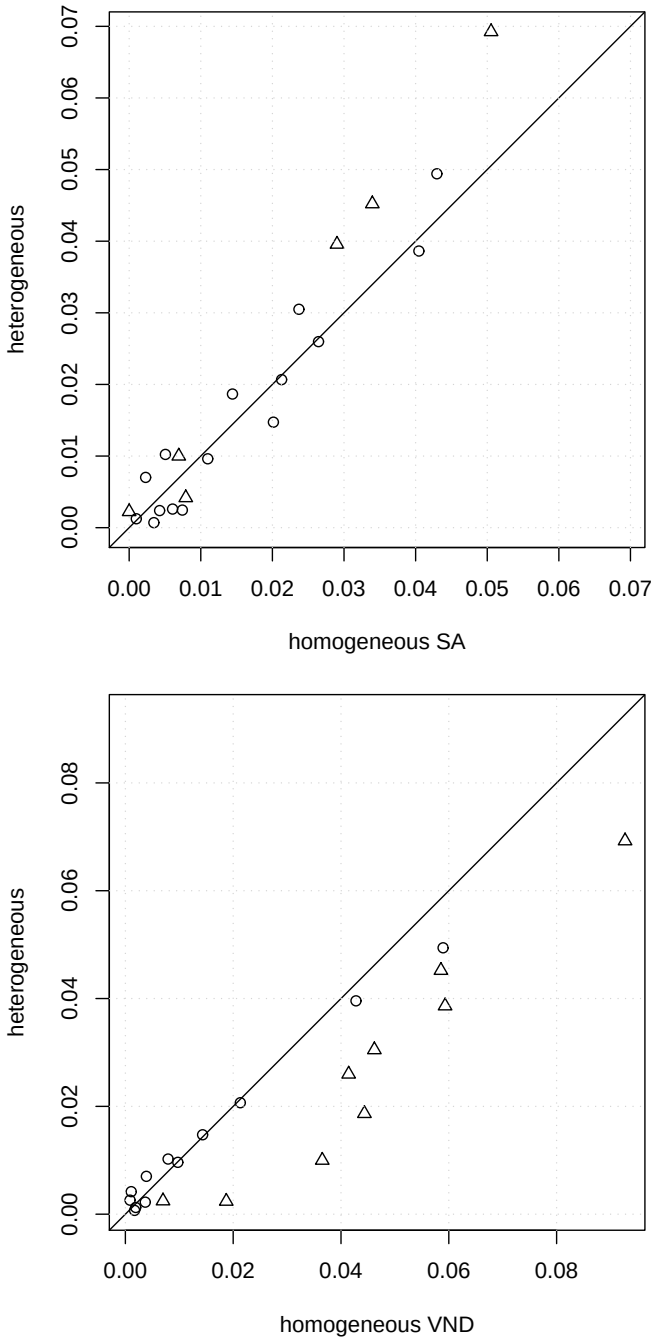


Figure 4.8: Heterogeneous vs. Homogeneous cooperation: comparison of the average deviation from the optimum (values are normalized). A triangle represents an instance where both algorithms perform differently at significance level 0.05 (Mann-Whitney U test).

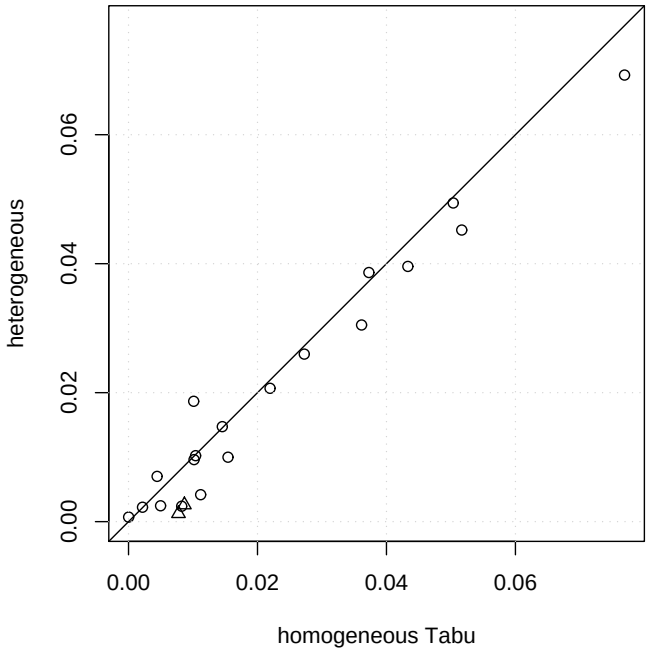


Figure 4.9: Heterogeneous vs. Homogeneous cooperation: comparison of the average deviation from the optimum (values are normalized). A triangle represents an instance where both algorithms perform differently at significance level 0.05 (Mann-Whitney U test).

axis.

A circle is used when the difference between both algorithms in the corresponding instance is not significant (the point lies very close to the diagonal). A triangle is used in other case. If the marker is above the diagonal, then the algorithm in X is better than the one in the Y axis.

When making the comparison between a cooperative strategy and an individual method, instead of providing three plots (one per each single method), we used the values of the best single metaheuristic. Of course, this best method is not the same for every instance, so the comparison target for the cooperative schemes is stricter.

These comparisons are shown in Figures 4.4 and 4.5. Just for the case of *H-VND*, one can see that there are individual strategies that achieved better error values. However, just in four cases, the differences have statistical significance. When the behaviour of the homogeneous cooperative strategies is compared, Figures 4.6 and 4.7, one can see that *H-SA* provides better results on a higher number of instances than *H-VND* and *H-Tabu*, being many of the improvements are significantly better (several triangles are shown). Also, *H-Tabu* provides better results than *H-VND* on a higher number of instances.

Finally, Figures 4.8 and 4.9 allow to compare the heterogeneous vs. homogeneous composition of the set of solvers in the cooperative strategy. The heterogeneous strategy achieves very similar results as *H-Tabu*, with many instances lying in the diagonal. When compared against *H-VND*, the results obtained are clearly better in almost every tested instance. Regarding the behaviour against *H-SA*, one can see that the number of points below and above the diagonal is roughly the same. However, when *H-SA* is better, the corresponding difference is larger (the point is more separated from the diagonal).

4.5 Analysing the influence of the cooperation scheme

The aim of our experimentation is to assess the benefits of the new control rule proposed and the performance of the strategy when uses different control rules. To this end, we will compare the following models:

- The independent strategy (I), which is the baseline case, where the

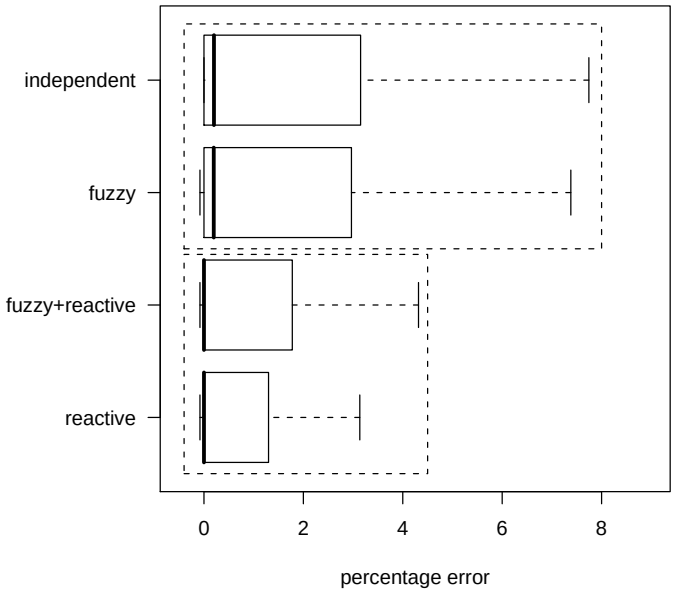


Figure 4.10: Box diagram for the percentage error obtained by I, F, R and F+R strategies over all instances. The algorithms grouped by a dotted rectangle do not perform differently at significance level 0.05 (non-parametric Mann-Whitney U-test)

solvers do not exchange information.

- The cooperative strategy that uses the Reactive rule (R).
- The cooperative strategy using the Fuzzy rule (F).
- The cooperative strategy using both rules (F+R).

Each strategy is run 30 times, starting from different initial random solutions, for each instance. In this case $freq\_interval$ was set to [500, 560]. The performance is assessed in terms of: quality of the solutions; convergence speed; and how the rules modifies the threads.

4.5.1 Benefits of the Cooperative-Reactive hybrid

Looking at the general picture, we show in Figure 4.10 a box diagram with information about the percentage error obtained by each method over all instances, where the boxes are sorted by the median. The Kruskal-Wallis non-parametric test for multiple comparisons has been used to asses the

differences between the performances of the four strategies. The null hypothesis could not be rejected at significance level 0.01. The statistical information about the comparison between pairs of algorithms is also shown in Figure 4.10: when two boxes are inside a dotted rectangle, a statistically significant difference could not be shown between the percentage error distributions of the corresponding algorithms (non-parametric Mann-Whitney U-test at significance level $\alpha = 0.05$). Looking at the medians there is no strong differences between the algorithms. However, if we take into account the upper quartile represented by the whisker, the differences are more noticeable, and show that the R strategy obtains the best performance. The second position is occupied by F+R, followed by F and I in that order. Furthermore, the statistical non-parametric test shows that R and F+R perform significantly better that the two worst methods.

The following analysis focuses on detailed per-instance results. Figures 4.11, 4.12 and 4.13 show a number of scatter plots providing pairwise comparisons among the four methods evaluated. Every instance is plotted as a pair (x, y) where x (y) is the normalised mean percentage error obtained by the strategy named in the X (Y) axis. If the marker is above the diagonal, then the algorithm in X is better than the one in the Y axis. A triangle is used when there is a statistically significant difference between the performance of the two algorithms for the corresponding instance (non-parametric Mann-Whitney U-test at significance level $\alpha < 0.05$). A circle is used in the other case.

The first plot shows how F hardly improves over I. Only in two instances the difference between both methods is statistically significant, which means that this rule does not obtain good results for this problem. When the coordinator uses R, it outperforms the I strategy. As we can see in the plot, the performance of this coordination type is significantly better on seventeen instances. When the control is carried out by F+R, the results are similar to those obtained by R, although now the number of cases where this control rule achieves a significant difference over I is reduced to fourteen.

If we compare the different rules among them, we observe that F+R improves upon F. The contribution of F+R is significantly evident in thirteen instances. However, when F+R is compared with R, apart from one instance,

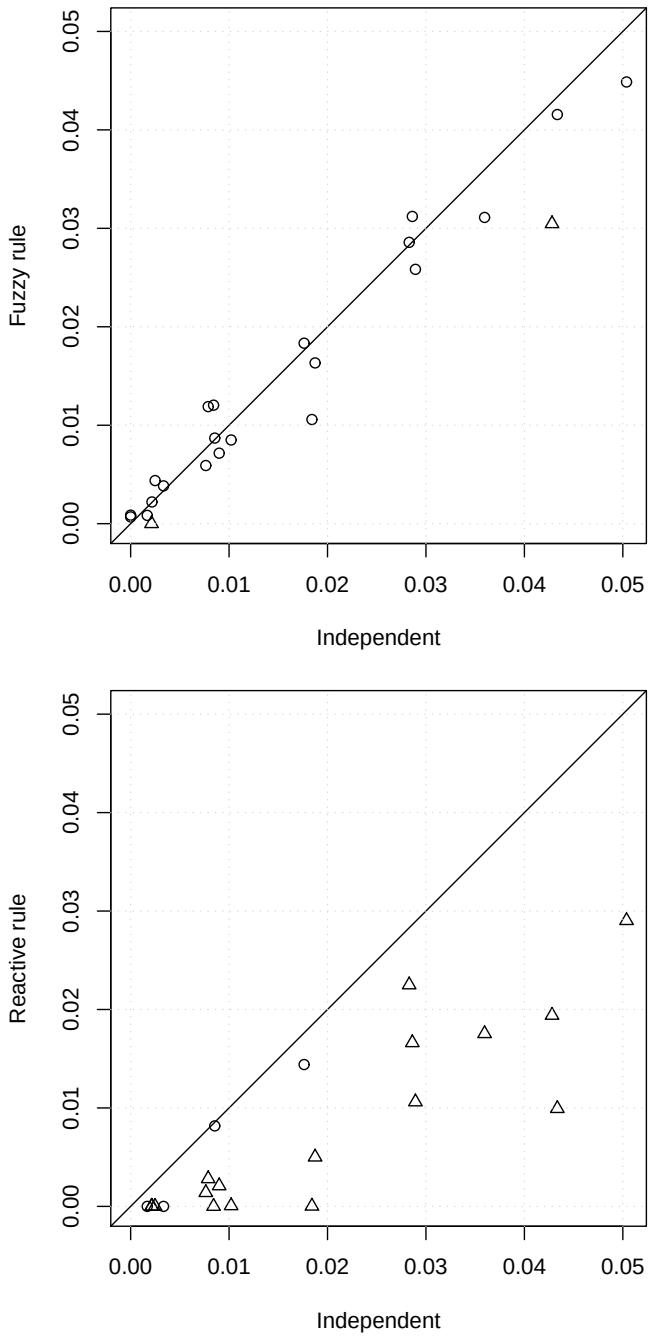


Figure 4.11: Comparison of the mean relative deviation from optimum (smaller values are better). Triangles represent the instances on which the algorithms being compared perform differently at significance level 0.05 (Wilcoxon's unpaired rank sum test).

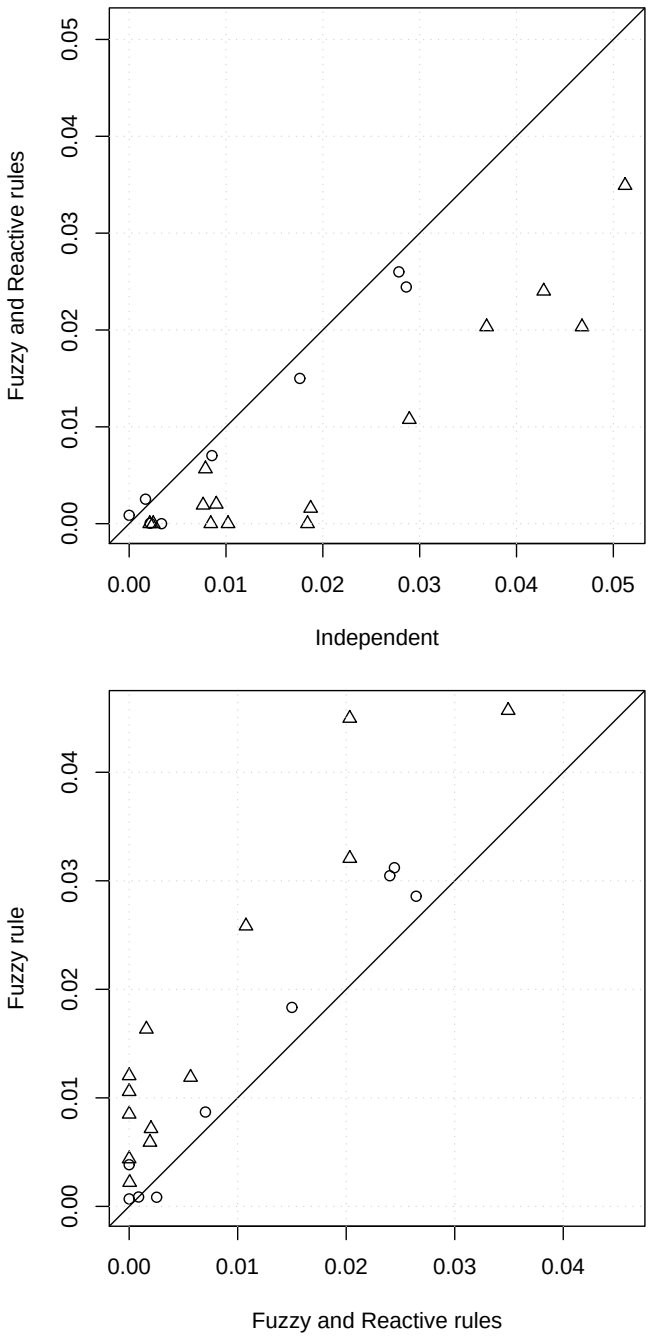


Figure 4.12: Comparison of the mean relative deviation from optimum (smaller values are better). Triangles represent the instances on which the algorithms being compared perform differently at significance level 0.05 (Wilcoxon's unpaired rank sum test).

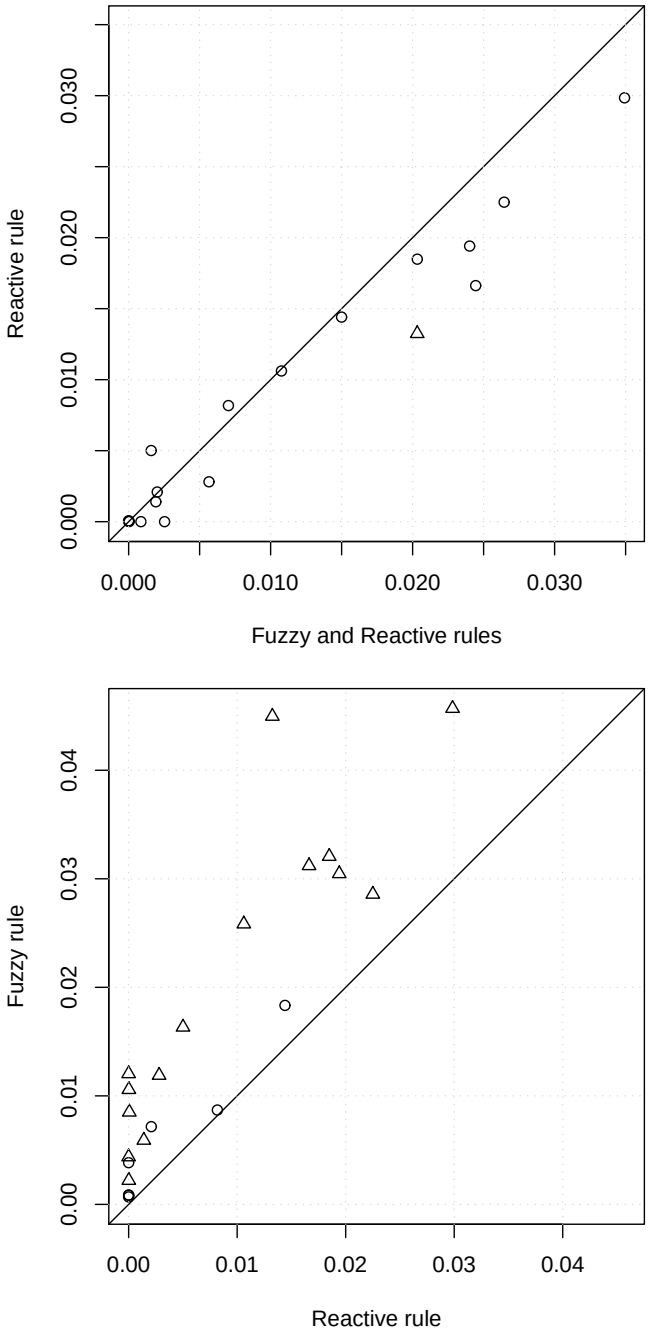


Figure 4.13: Comparison of the mean relative deviation from optimum (smaller values are better). Triangles represent the instances on which the algorithms being compared perform differently at significance level 0.05 (Wilcoxon's unpaired rank sum test).

the null hypothesis of the equivalence of the performances could not be rejected. The comparison between F and R confirms the intuition on the importance of the reactive rule, since on fifteen cases there is a significant improvement over F.

The next step is to identify the instances where there is a significant difference between the different cooperation schemes. Table 4.5 shows the mean percentage error for the different strategies as well as the std. deviation. From the table, we can notice that the differences of R and F+R with respect to F and I are larger when the size of the instances increases, i.e., for a fixed number of nodes, when the number of hubs increases. If we consider the best solution found, which can be seen in Table 4.6, something similar happens. The difference in terms of both the number of times the optimum is reached and the quality of the best solution found is bigger for instances with a higher number of hubs, with the only exception of F in 200-{20}.

4.5.2 Study of the dynamic behaviour

Two different aspects of the dynamic behaviour of the strategy will be investigated now. The first is the performance of the strategies during the search process. For this, we studied how the percentage error of the best solution found for each method evolves over time. We will focus on the six hardest instances, which are 100-{10,15,20} and 200-{10,15,20}. The results are shown in Figures 4.14, 4.15 and 4.16.

The first issue we want to highlight is the early stagnation in local minima experienced by I and F for the instances with one hundred nodes. This behaviour can also be observed when the instances have two hundred nodes, although in these cases it takes places in the last stages of the search. Unlike these methods, R and F+R do not suffer from this problem and are able to improve the quality of the solutions during the whole search process. Just on instance 100-{10}, a small stagnation is noticeable. This fact confirms that the reactive control rule is capable of driving the diversification of the strategy in such a way that the solvers can escape from the different local minima.

		mean				std. dev			
N	H	I	F	R	F+R	I	F	R	F+R
20	3	0.25	0.44	0.00	0.00	0.74	1.00	0.00	0.00
	4	1.87	1.63	0.50	0.16	1.54	1.58	1.02	0.53
	5	1.02	0.85	0.01	0.00	1.04	1.33	0.04	0.00
25	3	0.90	0.72	0.21	0.20	1.38	1.29	0.77	0.77
	4	0.76	0.59	0.14	0.19	1.26	1.26	0.17	0.17
	5	1.84	1.06	0.00	0.00	2.79	2.14	0.01	0.00
40	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	4	0.33	0.38	0.00	0.00	1.29	1.26	0.00	0.00
	5	0.79	1.19	0.28	0.57	1.08	1.18	0.77	1.02
50	3	0.17	0.08	0.00	0.25	0.64	0.46	0.00	0.77
	4	0.84	1.20	0.00	0.00	2.08	2.45	0.00	0.00
	5	2.86	3.12	1.66	2.44	2.05	2.29	2.11	2.16
100	3	0.00	0.09	0.00	0.09	0.00	0.47	0.00	0.47
	4	0.21	0.00	0.00	0.00	1.14	0.00	0.00	0.00
	5	1.76	1.83	1.44	1.50	0.81	0.76	1.11	1.08
	10	2.89	2.58	1.06	1.08	1.74	1.23	0.98	0.98
	15	5.12	4.94	1.75	2.47	1.56	1.61	1.16	1.44
	20	4.35	3.87	2.50	2.68	1.39	1.64	0.96	0.79
200	3	0.00	0.07	0.00	0.00	0.00	0.38	0.00	0.00
	4	0.22	0.22	0.00	0.00	1.13	1.13	0.01	0.01
	5	0.77	0.79	0.74	0.62	0.43	0.46	0.53	0.52
	10	2.94	2.97	2.36	2.76	1.18	1.12	1.39	1.68
	15	6.68	6.12	4.51	5.03	1.58	1.76	1.49	2.17
	20	6.81	5.54	4.41	4.88	1.69	2.16	1.13	1.82

Table 4.5: Mean and std deviation for I, F, R and F+R

		best solution			
N	H	I	F	R	F+R
20	3	25	23	30	30
	4	11	13	24	27
	5	10	16	29	30
25	3	20	21	27	28
	4	4	7	16	13
	5	11	12	29	30
40	3	30	30	30	30
	4	28	27	30	30
	5	7	5	24	21
50	3	28	29	30	27
	4	20	19	30	30
	5	8	8	18	12
100	3	30	29	30	29
	4	25	30	30	30
	5	2	2	11	10
	10	0.49	0.49	3	2
	15	1.48	1.27	0.75	0.42
	20	2.38	1.14	0.73	0.64
200	3	30	29	30	30
	4	20	16	28	26
	5	0.17	1	3	1
	10	1.66	1.30	0.11	0.15
	15	3.15	1.89	1.56	1.48
	20	3.24	2.42	2.77	2.82

Table 4.6: Best solution for I, F, R and F+R. In this table, when the value is an integer it indicates the number of times that the optimum was reached. Otherwise, the cost of the best solution found by that strategy

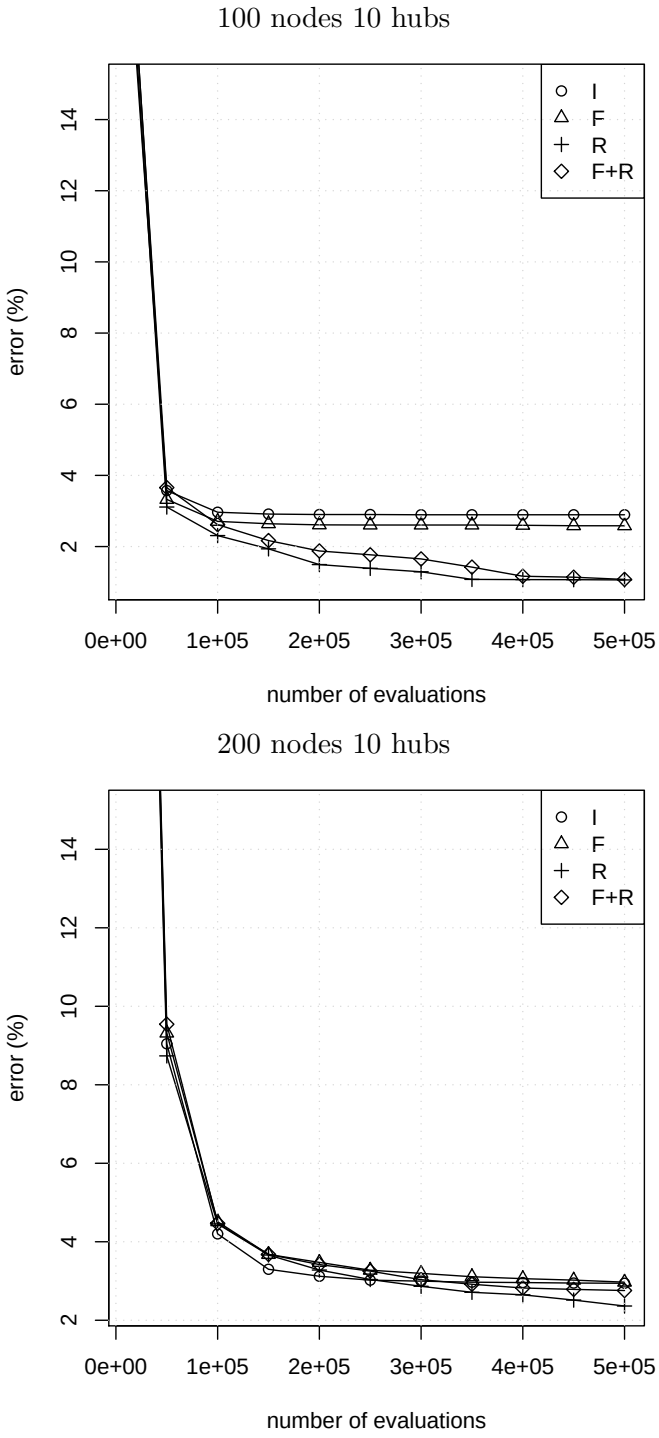


Figure 4.14: Evolution of the mean percentage error of the best solution found by I, F, R and F+R during the execution time

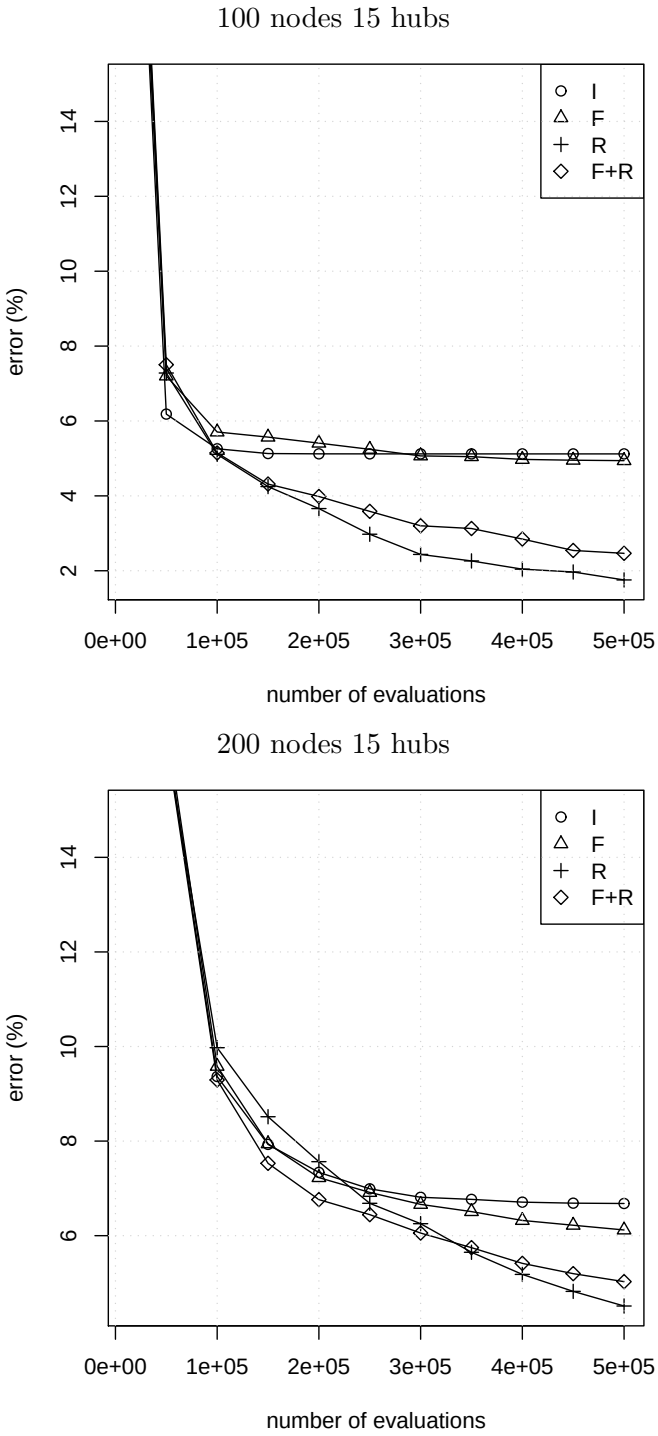


Figure 4.15: Evolution of the mean percentage error of the best solution found by I, F, R and F+R during the execution time

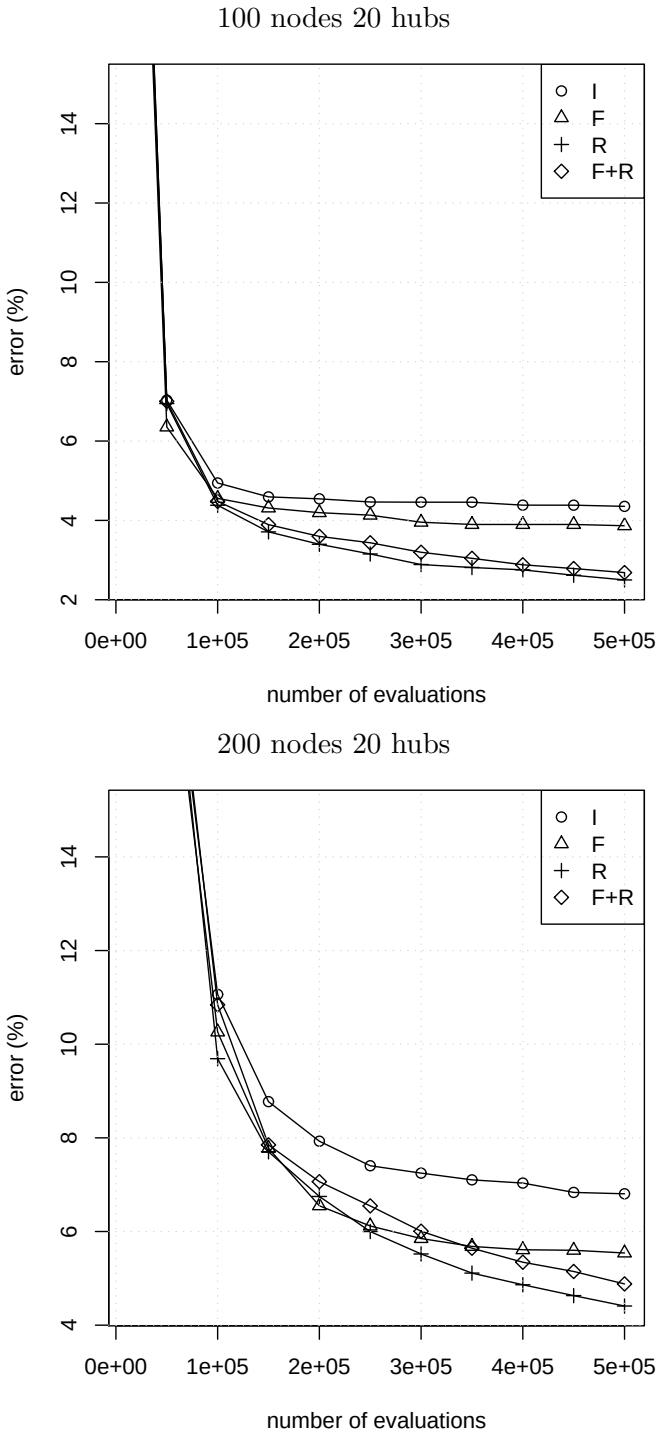


Figure 4.16: Evolution of the mean percentage error of the best solution found by I, F, R and F+R during the execution time

Another interesting behaviour can be found in 200-{20}. In this instance, F achieves a faster convergence than I, which leads to better results. This can be considered as an indication that the use of this control rule is useful for larger instances where the concentration of the solvers in the same region of the search space is needed to, at least, obtain high quality local optima. More experimentation is needed in order to confirm this fact.

The other aspect of the dynamic behaviour we have studied is the evolution of the rule activation with time. Figures 4.17, 4.18 and 4.19 show the evolution of the mean number of times the control rule is fired for I, F, R and F+R. For this last one, the disaggregated behaviour for each rule (F+R:F and F+R:R) is shown. As in the former case, we focus on the six hardest instances.

Looking at those figures for instance with 100 nodes, we can observe a much higher number of activations of R and F+R:R with respect to F and F+R:F. However, that order is inverted when the number of nodes is increased to 200. This effect is due to an important drop in the number of triggers for R and F+R:R. This behaviour variation is explained by the different performance of the solvers in the two cases. As we have seen before, the independent strategy stagnates in the first phases of the search process for instances with one hundred nodes. This situation is captured by the reactive rule, which increases the number of triggers since the early stages of the search. On the contrary, for 200-10,15,20, the independent solvers experience a slower convergence, and stop improving at the end of the process. This is also reflected in the behaviour of the reactive rule, that delays its activation. In this way, we see that the reactive rule is able to detect the difference between both instance sizes, adjusting its behaviour to each case.

We can also observe that using the two rules simultaneously produces an increase in their number of activations. This is due to the difference between the objectives of both rules. Since the fuzzy rule tries to reallocate the threads around promising regions of the search space, the probability of solvers to find big local minima is higher, which leads the reactive rule to be triggered more times. Although in a lesser degree, that interaction in the opposite sense also happens. When the cooperation is driven by the

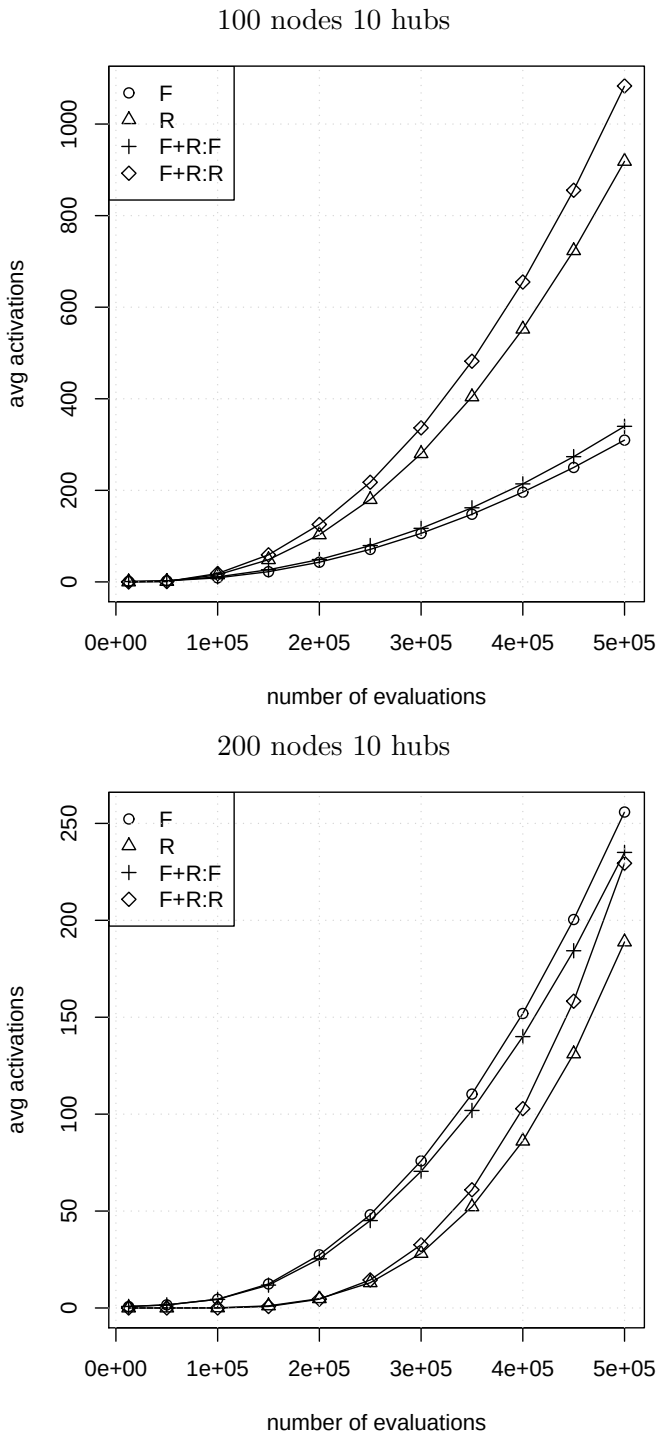


Figure 4.17: Evolution of the mean number of times the control rule is fired for I, F, R and F+R. For this last strategy, the disaggregated behaviour for each rule (F+R:F and F+R:R) is shown

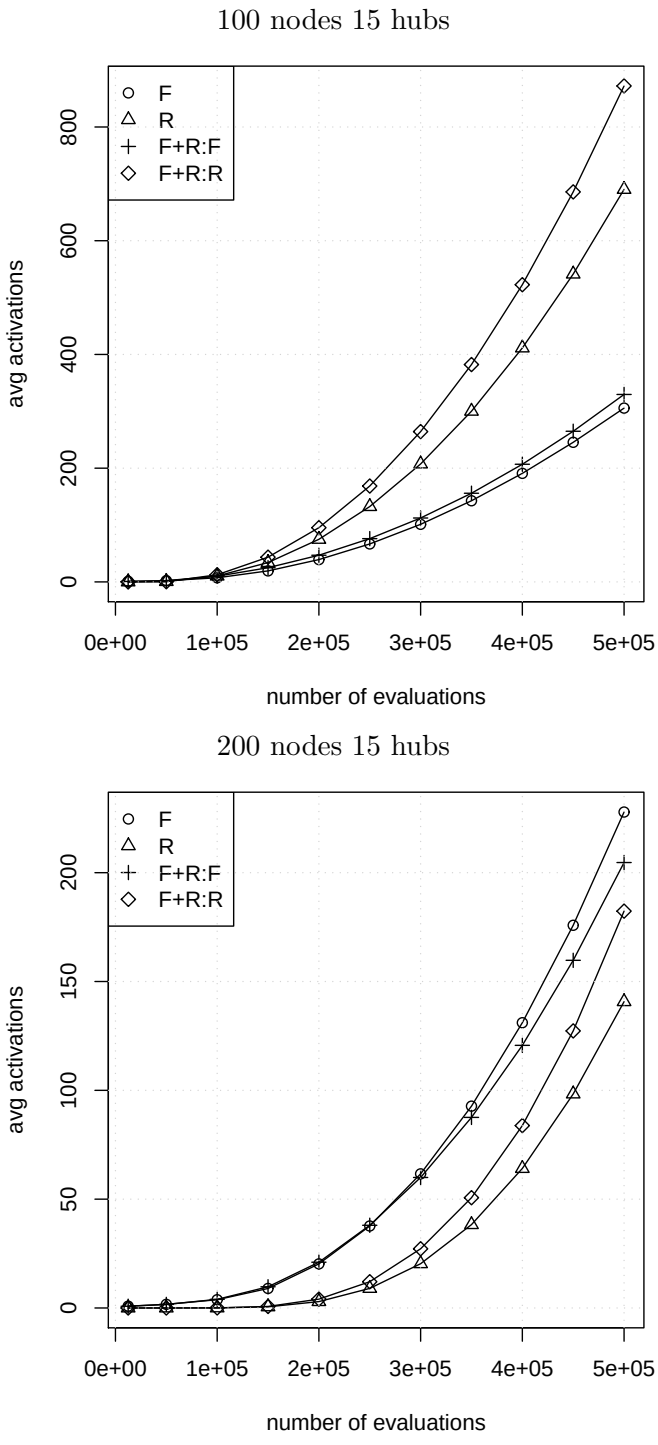


Figure 4.18: Evolution of the mean number of times the control rule is fired for I, F, R and F+R. For this last strategy, the disaggregated behaviour for each rule (F+R:F and F+R:R) is shown

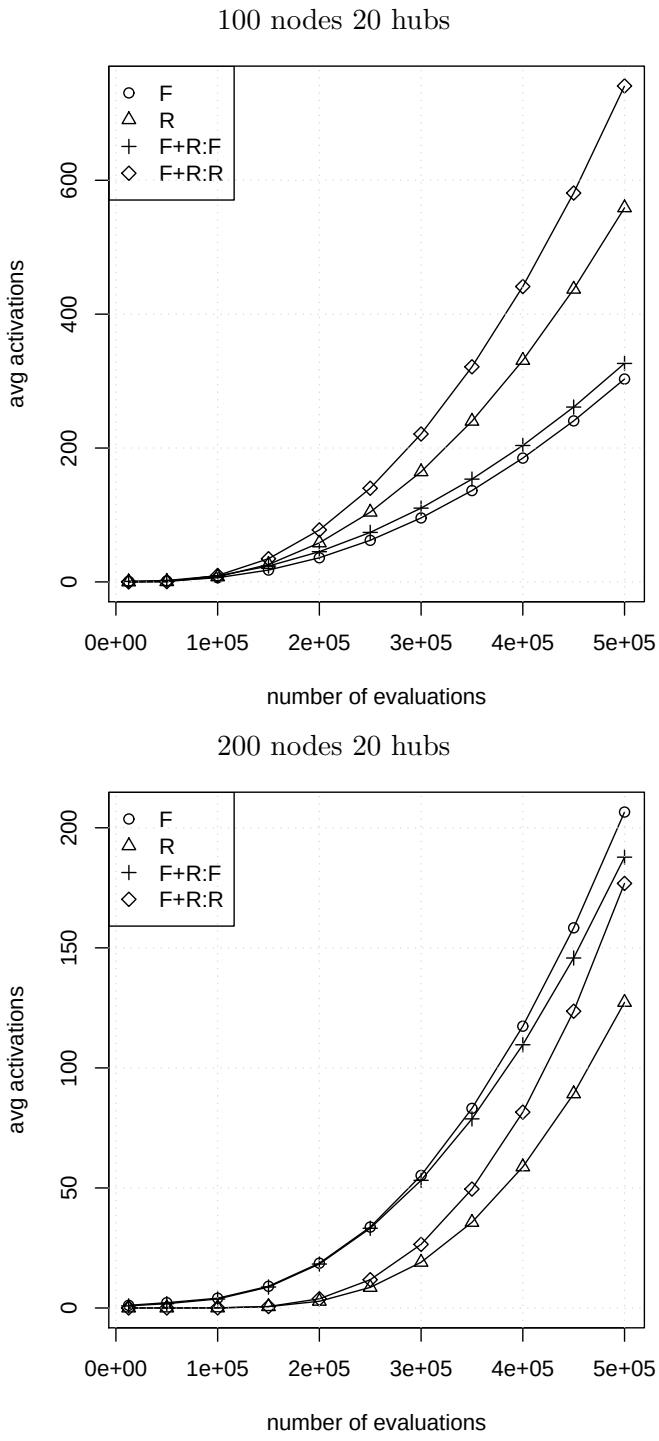


Figure 4.19: Evolution of the mean number of times the control rule is fired for I, F, R and F+R. For this last strategy, the disaggregated behaviour for each rule (F+R:F and F+R:R) is shown

Reactive rule, the coordinator attempts to bring the solvers away from the local minima, and put them in regions of the search space relatively far from the best solution. If this distance is big enough and the solver does not achieve a good improvement rate, then the fuzzy rule will be fired for this thread.

4.6 Conclusions

This chapter has focused on an in-depth study of centralised cooperative strategies, departing from the research done on a method where the set of search agents are controlled by a central coordinator that takes decisions basing on a fuzzy rule. We have extended this research in two key points. In first place we have studied how the definition of the solvers can influence the performance of the cooperative strategy. In other words, using as test bed the Uncapacitated Single Allocation p-Hub Median Problem (USApHMP) we have analysed the behaviour of the method when all solvers implement the same heuristic (homogeneous strategy) and when each solver implements a different one (heterogeneous strategy). To have a reference of the performance of such strategies, these have been compared with the isolated individual version of the distinct heuristics used as solvers.

Concerning this last point, we saw that by means of homogeneous cooperation based on our schema, the average error of three different metaheuristics can be markedly reduced.

We also compared the different cooperative strategies studied versus the best individual method which can vary from an instance to another. The obtained results have shown that cooperation, both homogeneous one and heterogeneous one, leads to equal to or better average fitness values than the best individual metaheuristic in virtually all cases.

The last aspect analysed here was the existing performance differences between the cooperative methods, where it should be highlight that a) those homogeneous strategies whose solvers implement the best global individual metaheuristic represent the best alternative in terms of performance and b) when the comparison is done between the heterogeneous and the homogeneous composition, we checked that the first one presents some advantages over the second ones.

The second point in which we extended the mentioned research was

the cooperation scheme. Concretely, we have incorporated a new control rule based on the reactive search framework into the centralised cooperative strategy. The experiments, also done on instances of the USApHMP, have showed that the proposed reactive cooperation scheme achieves better results with respect to a strategy based on independent solvers (where solvers do not exchange information), and with respect to the same cooperative strategy based on a fuzzy control rule previously proposed. We also tested the cooperative strategy using both rules, reactive and fuzzy, at the same time. In this case, the bigger complexity did not pay of because the reactive rule alone performs at least as well as both rules together. Moreover, the reactive rule is able to adapt its behaviour according to the characteristics of the instance, and is effective for detecting stagnation and for driving diversification strategies.

The behavior of the cooperative strategy coupled with the fuzzy rule is somehow deceptive as it can not perform better than the independent strategy. Further research is needed in order to assess if this is strictly related with the p-hub problem, since previous works showed the benefit of such combination [40, 125].

Chapter 5

New applications for centralised cooperative strategies: Dynamic Optimisation Problems

This chapter focuses on the application of centralized cooperative strategies to a new area, Dynamic Optimization Problems (DOPs), where they have not been tested before. The strategy presented in the former chapter is adapted and provided with trajectory-based solvers to deal with this type of problems. Two state-of the art methods are considered for comparison purposes: multi-QPSO and Agents. The main goals are: firstly, to draw attention on centralised cooperation schemes for DOPs; and secondly, to assess the possibilities of trajectory methods in the context of DOPs. The results show how the proposed strategy can consistently outperform the results of the two other methods.

5.1 Motivation

Recently, there is a growing research interest on the resolution of Dynamic Optimization Problems (DOPs) [16,20,21], due to its closeness to real-world situations (trade market prediction, metheorological forecast, robotics motion control, etc). The methods designed to deal with these problems should take into account that both the problem and the solution may evolve in time,

and besides, that it is more important to follow the optima as close as possible than to find the their exact position. From the optimization point of view, the most basic approach to dynamism is to simply restart the search whenever a change is detected, just as if it were a different problem. However, since the evolution in time usually occurs in a gradual way, facing each change as if it were a completely new problem implies that important information gathered in previous resolutions is wasted. The reuse of this information allows for rapid adaptation when a change in the problem occurs, thus decreasing the time required to find a new solution. This is necesary in many practical cases, where there is no time for performing exhaustive explorations of the search space.

There are already some methods to deal with DOPs, but there are not yet clear criteria about which method is better to apply or how to face a new dynamic problem. The most commonly used methods have been evolutionary algorithms, but recently, other approaches have been applied and shown to outperform them in some cases ([16]). Particle Swarm Optimization (PSO)) is among the most competitive methods that have been applied so far to dynamic optimization. In particular, the PSO variant with multiple swarms and quantum and trajectory particles (multi-QPSO, [16]) has obtained very good results on the most commonly used test problems.

Most of the methods used, including PSO and evolutionary algorithms, are population-based methods where the use of several solutions helps to avoid local minima and to better detect when a change on the problem occurs. This statement has been quite assumed and most research done in DOPs focuses on the recommended populational methods with little attention put on the possibility to use trajectory-based methods. Another less discussed aspect of the methods for DOPs is that most of them show some form of cooperation. In evolutionary algorithms, for example genetic algorithms, the evolution of the populations can be seen as a form of implicit cooperation among their individuals. Moreover, the multi-QPSO method shows cooperation among particles of the same swarm whereas, at the same time, there is competition between the different swarms. Despite that, as far as we know, little focus has been put on the cooperative aspect of the methods, as well as the cooperative behavior, not being mentioned or considered in the development of new methods. Moreover, since the cooperation present on the most used methods is implicit, it is not clear if it is being an

important factor for achieving a good performance or not.

Taking into account that trajectory-based methods have been rarely considered in DOPs and the indications of the good behaviour showed by some cooperation forms we decided to assess the possibilities of trajectory methods coupled with a cooperation scheme in this context, where populational methods have traditionally been the recommended option. Concretely, we focus on DOPs with a continuous search space and where the element that change over the time is the objective function. To this end, we used again the same centralised cooperative strategy showed in chapter 4 adapting it in first place to continuous problems and subsequently to DOPs.

Another objective of this chapter is to test if a proper use of trajectory methods coupled with a cooperation scheme can lead to competitive results respect to the state of the art methods in DOPs [16]. Two additional methods were considered for comparison purposes. The first method is the multi-QPSO method discussed before that is known to be among the state of the art methods for DOPs and that will be used as a baseline for comparison. The second method is an explicit decentralized cooperation scheme presented on [122, 123] where multiple agents cooperate to improve a set of solutions stored on a grid.

This chapter is structured as follows. Section 5.2 shows the different adaptations done to the centralised cooperative strategy to deal with both continuous search spaces and DOPs as well as the method implemented by the solvers. In 5.3, the benchmarks used for the experimentation, the two compared methods and the implementation details are described. Then, Section 5.4 focus on the experiments done to test the performance of the methods and on the analysis of the results obtained. Finally, the conclusions are presented in Section 5.5.

5.2 Description of the centralised cooperative strategy for DOPs

In this section we will describe the different modifications done to the centralised cooperative strategy presented in chapter 4 in order to make it suitable for both continuous search spaces and DOPs.

5.2.1 From combinatorial to continuous search spaces

Before describing the changes, let us overview the whole working of the strategy. As we saw, the strategy consists on a set of solvers/threads to deal with the problem at hand. The coordinator processes the information received from them and produces subsequent adjustments of their behaviour by sending "orders". To achieve this exchange of data, a blackboard model was used. After an initialization stage, the solvers are executed asynchronously while sending and receiving information. The coordinator checks which solver provided new information and decides whether its behaviour needs to be adapted using the rule base. If this is the case, it will calculate a new behaviour which will be sent to the solvers.

The reports that solvers send to the coordinator contains the next items:

- Solver identification
- A time stamp
- the current solution of the solver at that time
- The best solution reach until that time for this solver
- A list with the local minima found by the method since the last report

The coordinator stores the last two reports from each solver that uses to calculate its improvement rate. This value together with the cost of the current solution are stored in two memories. The list of local minima is processed by the coordinator that keeps the history of all local optima in a hash table. Each entry of this table has also a collision counter with the number of times that a solution has been visited by any search thread.

It is in this hash table where one of the modification applied is found. Since in continuous space two distinct but very close solutions can be within the same local minima, if we use the former approach we would consider two different optima. To solve this problem, the coordinator keeps the history of all local optima found by the solvers in a memory denominated Visited Region List (VRL). Now, the local minima are considered as the regions defined by a hypersphere with radius ρ and centre in the points stored in the VRL. As in the former case, each entry of the VRL also maintains a register *cc* (collision counter) with the frequency of visiting that region by any search thread.

The reactive rule is used to coordinate the solvers although we take the consequent of the fuzzy rule. This change is not related to the adaptation to continuous problems but the simplicity of the strategy. Therefore, the rule is the following:

> **if** *cc* of the last local minimum visited by $solver_i$ is bigger than $\lambda_{reaction}$, **then** the coordinator sends C_{best} with a slight modification to $solver_i$.

C_{best} is also taken by the solvers as its new current solution, starting the search trajectory from this point.

5.2.2 From static to dynamic problems

The main issues to deal with when a centralised cooperative strategy should be adapted from static to this type of dynamic problems are to check when the fitness function has changed, on one hand, and restart those memories that does not contain relevant information for the new search space, on the other hand. These issues are handled in both the coordinator and solver sides. Next we will explain the steps followed in each case:

- **Solver side:** The mechanism that detects fitness function changes is implemented in solver's side. The solver should keep a list with all or recent local minima found. Then, before starting to explore the neighbourhood of the current solution, the solver reevaluates the stored local optima and checks if the fitness has changed. If so, it stops the reevaluation and the state of the local solver is restarted (the current solution and the list of local minima are reevaluated; tabu lists, memories, counters, etc. are reset). Apart from this, solver keeps a register, that we call FCC, with the number of times that the fitness function has changed. The current value of this counter is sent by the solvers to the coordinator in every report.

- **Coordinator side:** The coordinator also needs to know when the fitness function changed to restart its state. For this purpose, the coordinator keeps a counter $CFCC$ that is initialised to zero. For each received report, it checks if the FCC sent by the solver is higher than $CFCC$. If so, it determines that a new fitness function change has happened and updates $CFCC$ to this value. It reinitialises all its

memories and registers, excepting, of course, this counter. In the case that the FCC is lower than $CFCC$, the coordinator notifies to the solver through a report that a fitness function change has taken place. This makes the solver uptades its FCC and restarts its state as we explained before. IF FCC is equal to $CFCC$, the coordinator keeps on with its normal working.

5.2.3 Description of the method implemented by the solvers

The method implemented by the solvers is a tabu search algorithm that can be considered as a hybridization of two metaheuristics previously presented in literature, DOPE [58] and DTS [77]. From the first one we have taken the variable move step as well as some parts of its tabu neighbourhood exploration method. From the second one, we have introduced its diversification procedure.

Before starting with the description of the tabu search, we should point out an important issue to determine in continuous optimization when we search better solutions around a specific point: the size of the movement. A small step size can lead to an important waste of objective function evaluations, whereas a big movement length make difficult find solutions with enough accuracy. For this reason it is interesting to use a large step size at the beginning of the search to do a better exploration of the search space, then it is possible to reduce the size to obtain a better accuracy of the solution. In this way, in our algorithm, we start with a step size of length δ_{init} and every time the exploration of the neighbourhood of the current solution does not lead to a better one, this size is halved. When two consecutive steps get to improve the current solution the step size is multiplied by two. The length of the movement is delimited by the interval $[res, \delta_{init}]$, where res is a parameter which determines the precision of the algorithm.

The pseudocode for this method is showed in Algorithm 14. Tabu search starts with an initialization stage and after that, we find a loop which is repeated until the stop condition is fulfilled. Firstly, this loop checks if the step size has been reduced to a value lower than res. If so, it determines that the current solution is a local optimum and checks if it is near to a previously visited local minimum (the distance to a local optimum is lower than LRR) or not. To escape from it, the method applies a diversification

Algorithm 14 Continuous Tabu Search pseudocode

procedure *ContinuousTabuSearch*

$\delta \leftarrow \delta_{init}$
$x \leftarrow$ **GenerateInitialSolution()**
$x_{best} \leftarrow x$
$f_{best} \leftarrow f(x_{best})$
$consAdvances \leftarrow 0$

while not stopping condition do
  if $\delta < res$ **then**
    $\delta \leftarrow \delta_{init}$

    if IsNearLocalMinimum(x_{best}) **then**
      {If x_{best} **is near to a local optimum, we apply the diversification procedure}**
      $x \leftarrow$ **Diversification**(x_{best})
      $x_{best} \leftarrow x$
      $f_{best} \leftarrow f(x_{best})$
      $consAdvances \leftarrow 0$
    else
      {x_{best} **is considered as a new local minimum}**
      $acceptNoImp \leftarrow maxNoImp$
      AddLocalMinimum(x_{best})
      $f_{best} \leftarrow \infty$
      $consAdvances \leftarrow 0$
    end if
  end if

  $x \leftarrow$ **ExploreNeighbourhood**(x,$acceptNoImp$,δ)

  if $f(x) < f_{best}$ **then**
    $x_{best} \leftarrow x$
    $consAdvances \leftarrow consAdvances + 1$
    if $consAdvances = 2$ **then**
      $\delta \leftarrow$ **IncreaseStep**(δ)
      $consAdvances \leftarrow 0$
    end if
  else
    $\delta \leftarrow$ **ReduceStep**(δ)
    $acceptNoImp \leftarrow acceptNoImp - 1$
  end if
end while

Algorithm 15 NeighbourhoodExploration(x,$acceptNoImp$, δ)

procedure *NeighbourhoodExploration*

$x_{new} \leftarrow$ **OptimisticMovement(**x**,** $lastMovement$**,** δ**)**
if $f(x_{new}) < f(x)$ **then**
  return x_{new}
end if
$x_{new} \leftarrow$ **ADD(x,** δ**)**
if $f(x_{new}) < f(x)$ **then**
  return x_{new}
end if
$x_{new} \leftarrow$ **TabuExploration(x,** δ**)**
if $acceptNonImp = 0$ **AND** $f(x_{new}) > f(x)$ **then**
  return x
else
  return x_{new}
end if

procedure [1], to restart the search in a non explored region, or allow a certain number of non improvement movements, respectively.

After this, the method explores the neighbourhood and checks if the chosen neighbour is better than the best solution seen until now. In this case, the search is allocated at this point and the step size is increased if two consecutive improvements has taken place. Otherwise, it is halved.

One of the main parts of this algorithm is the way in which it explores the neighbourhood. Concretely, this method considers three different exploration modes that are tested in a specific order. If a determined mode does not lead to a successful movement, then the next one is tried. The three exploration forms are described below:

1. **Optimistic search**: In first place our algorithm tries a step in the last good direction, that is, the direction of movement chosen in the previous neighbourhood exploration, since we can hope that this trajectory will be still good in the next iteration.

2. **Approximate descent direction (ADD)**: If the last procedure does

[1]The diversification movement is done following the procedure 2.1 of [77]

not lead to an improvement, then the heuristic attempt to find a good descent direction using the ADD method [76].

3. **Tabu exploration**: In the case that the two former movements do not work, the algorithm carries out a more exhaustive search within the neighbourhood generating $2n$ neighbours. Concretely, for each one of n vectors that compose the basis of the vectorial space defined by the problem, we take two solutions in this direction, one in positive sense and one in negative.Then, the best non-tabu move or the best tabu move, if it fulfills the aspiration level, is taken as new current solution of the method. The tabu list is composed of the reverse of moves previously accepted, and a move becomes non-tabu after a given number of iterations defined by the parameter *tenure*.

For a better understanding of the neighbourhood exploration, its pseudocode is given in Algorithm 15.

5.2.4 Classifying the strategy

The classification of this strategy respect to the three taxonomies sawn in Section 3.3 is roughly the same as the one we stated in the former chapter, excepting that in this case we only deal with an heterogeneous strategy. This fact only alters the classification in Talbi's flat taxonomy (heterogeneous nature) and the dimension *search differentiation* (MPDS) in the one of Crainic et al.

5.3 Experimental framework

In this part of the chapter, we will show the instances of DOPs used as benchmarks. Afterwards, the two state of the art methods, that will be compared with the centralised cooperative strategy, are described. Next subsection is devoted to explain the tabu search used by the solvers and finally, some others implementation details will be seen.

5.3.1 Problems

The problems used to test our strategy are the Moving Peaks Benchmark (MPB) and three commonly used real test functions (Ackley, Griewank and Rastrigin). Their description is given below.

The Moving Peaks Benchmark

The Moving Peaks Benchmark (MPB) is a test benchmark for DOP's originally proposed in [20]. Its current version, and the most updated source of information about it, is available at the webpage:

http://www.aifb.uni-karlsruhe.de/~jbr/MovPeaks/

The MPB has become a de facto standard for analysing the performance of any new DOP methods and it has been chosen as the first problem to tackle for the purposes of this chapter.

Informally, the MPB consists on a set of **m** peaks, each of which has its own height (**h**), width (**w**), and location ($\overrightarrow{\mathbf{p}}$). The shape function of each peak i is defined as:

$$\mathbf{f}_i(\overrightarrow{x}) = \mathbf{h}_i - \mathbf{w}_i \sqrt{\sum_j (x_j - \mathbf{p}_j)^2} \tag{5.1}$$

and the fitness function of the problem is defined as the composition of all the peak functions, which, since it is a maximization problem, is as follows:

$$\mathbf{F}(\overrightarrow{x}) = max\{\mathbf{f}_i(\overrightarrow{x}) : \forall i\} \tag{5.2}$$

Changes to a single peak are described by the following expressions:

$$\mathbf{h}_i(t+1) = \mathbf{h}_i(t) + \mathbf{h}_{severity} \cdot \mathbf{N}(0,1) \tag{5.3}$$

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \mathbf{w}_{severity} \cdot \mathbf{N}(0,1) \tag{5.4}$$

$$\overrightarrow{\mathbf{p}}_i(t+1) = \overrightarrow{\mathbf{p}}_i(t) + \overrightarrow{\mathbf{v}}_i(t+1) \tag{5.5}$$

where the shift vector $\overrightarrow{\mathbf{v}}_i(t+1)$ is a linear combination of a random vector $\overrightarrow{\mathbf{r}}$ and the previous shift vector $\overrightarrow{\mathbf{v}}_i(t)$, normalized to length **s**, i.e.:

$$\overrightarrow{\mathbf{v}}_i(t+1) = \frac{\mathbf{s}}{|\overrightarrow{\mathbf{r}} + \overrightarrow{\mathbf{v}}_i(t)|}((1-\lambda)\overrightarrow{\mathbf{r}} + \lambda\overrightarrow{\mathbf{v}}_i(t)) \tag{5.6}$$

The random vector $\overrightarrow{\mathbf{r}}$ is created by drawing random numbers for each dimension and normalizing its length to **s**. Parameter **s** thus indicates the distance that a single peak moves when a change in the environment occurs (shift distance). Parameter λ indicates the linear correlation with respect to the previous shift, where a value of 1 indicates "total correlation" and a value of 0 "pure randomness". Finally, parameters $h_{severity}$ and $w_{severity}$

Table 5.1: Standard settings for the Scenario 2 of the Moving Peaks Benchmark

Parameter	**Value**
Number of peaks (m)	$\in [10, 200]$
Number of dimensions (d)	5
Peaks heights (h_i)	$\in [30, 70]$
Peaks widths (w_i)	$\in [1, 12]$
Change frequency (Δe)	5000
Height severity (h_s)	7.0
Width severity (w_s)	1.0
Shift distance (s)	$\in [0.0, 3.0]$
Correlation coefficient (λ)	$\in [0.0, 1.0]$

indicate the magnitude of the change to h and w respectively, in the units of those parameters.

One of the most used configurations for the MPB as a test suite is Scenario 2, which is described in the web page of the MPB, and consists on the set of parameters indicated in table 5.1. However, in order to reduce the number of variables to control in the experiments performed, neither h nor w were changed, and only the peaks position was varied (this is equivalent to choose $h = w = 0$).

Continuous Test Functions

Three commonly used multimodal real test functions have also been selected:

- **Ackley**:

$$f_{Ackley}(x) = -20\exp\left(-0.2 \times \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)\right) + 20 + e \tag{5.7}$$

- **Griewank**:

$$f_{Griewank}(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{5.8}$$

- **Rastrigin**:

$$f_{Rastrigin}(x) = \sum_{i=1}^{n} \left(x_i^2 - 3 \times cos(2\pi \cdot x_i)\right) + 3n \tag{5.9}$$

All these functions correspond to $n-$dimensional minimization problems whose optimal value is at $x = \vec{0}$. Unlike the MPB, where the number of local optima is equal to the number of peaks, these functions are highly multi-modal, which means that algorithms are more likely to get stuck in sub-optimal points, and therefore, are harder to optimize functions. Additionally, several functions of the same type can be combined to produce an even harder problem, as it has been done in the experiments conducted (section 5.4).

The dynamic behaviour is obtained by shifting position and height of the optimal with the same rules as in the MPB, that is:

$$\mathbf{h}_i(t+1) = \mathbf{h}_i(t) + \mathbf{h}_{severity} \cdot \mathbf{N}(0,1) \tag{5.10}$$

$$\overrightarrow{\mathbf{p}}_i(t+1) = \overrightarrow{\mathbf{p}}_i(t) + \overrightarrow{\mathbf{v}}_i(t+1) \tag{5.11}$$

where elements are defined as in the case of the MPB, using Eq. 5.3, 5.5, and 5.6.

Instead of modifying the functions f_{Ackley}, $f_{Griewank}$, and $f_{Rastrigin}$ to contain $h_i(t)$ and $p_i(t)$, points x and fitness function f are transformed according to Alg. 16.

Algorithm 16 Fitness evaluation of functions Ackley, Griewank and Rastrigin.

procedure $FitnessFunction(x, t)$

1: **Calculate** $h_i(t)$
2: **Calculate** $p_i(t)$
3: $x' \leftarrow x - p_i$
4: $fitness \leftarrow f(x')$
5: $fitness' \leftarrow fitness + h_i$
6: **Return** $fitness'$

5.3.2 State of the art methods

This section presents two methods. Firstly, we will describe the multi-QPSO algorithm with multiple swarms and quantum and trajectory particles that

will serve as a baseline for the comparison. It is a method that have been frequently used for DOPs and whose level of performance is known to be among the best ones. It can be seen as a population-based method with implicit cooperation.

Secondly, we will describe the Agents algorithm. It is also a populatio-based method where a set of agents cooperate to improve the solutions stored in a grid. Therefore, its behaviour can be seen as an explicit decentralized cooperation scheme.

The two methods are now presented in the following subsections.

Multi-QPSO method

The multi-QPSO is a variant of the PSO that was firstly introduced in [16] by Blackwell and Branke, which is specifically aimed at solving DOPs.

A swarm in the multi-QPSO is formed by two types of particles:

- *Trajectory* particles (also known as *classical* or *neutral*). These are the particles used by the canonical PSO algorithm, which positions are updated following the usual movement equations:

$$\mathbf{a}(t+1) = \chi[\eta_1 c_1 \cdot (\mathbf{x}_{pbest} - \mathbf{x}(t)) + \eta_2 c_2 \cdot (\mathbf{x}_{gbest} - \mathbf{x}(t))] - (1-\chi)\mathbf{v}(t) \quad (5.12)$$

$$\mathbf{v}(t+1) = \mathbf{v}(t) + \mathbf{a}(t+1) \quad (5.13)$$

$$\mathbf{x}(t+1) = \mathbf{x}(t) + \mathbf{v}(t+1) \quad (5.14)$$

  where $\mathbf{x}$, $\mathbf{v}$ and $\mathbf{a}$ are position, velocity and acceleration respectively. $\mathbf{x}_{pbest}$ is the best so far position discovered by each particle, and $\mathbf{x}_{gbest}$ is the best so far position discovered by the whole swarm. Parameters $\eta_1, \eta_2 > 2$ are spring constants, and c_1 and c_2 are random numbers in the interval $[0.0, 1.0]$. Since particle movement must progressively contract in order to converge, a constriction factor χ, $\chi < 1$ is used, as defined by Clerc and Kennedy in [30]. This factor replaces other slowing approaches in the literature, such as inertial weight and velocity clamping [48].

- *Quantum* particles. These particles were introduced in the multi-QPSO algorithm, and aim at reaching a higher level of diversity by

Algorithm 17 Pseudocode of Agents method.

```
procedure AgentsMethod()
  Initialise Matrix of Solutions randomly
  Distribute Agents in the Matrix
  while not stopping condition do
    Move Agents
    Change Solutions
    Evaluate Solutions
    Update solutions
  end while
```

moving randomly within a hypersphere of radius $\mathbf{r}$ centered on $\mathbf{x}_{gbest}$. This random movement is performed according to a probability distribution over the hypersphere, in this case, a uniform distribution:

$$\mathbf{x}(t+1) \quad = \quad \mathbf{rand}_{hypersphere}(\mathbf{x}_{gbest}, \mathbf{r}) \tag{5.15}$$

The general idea of the multi-QPSO is to use a set of multiple swarms that simultaneously explore the search space. This multi-swarm approach has the purpose of maintaining the diversity, in addition to the use of quantum particles. This is a key point for DOPs, since the optimum can change at any time, and the algorithm must be able to react and find a new optimum. In order to prevent several swarms from competing over the same area, an inter-swarm exclusion mechanism is also used, randomizing the worst swarm whenever two of them are closer than a certain distance. Intra-swarm cooperation and inter-swarm competition are the two basic forms of implicit cooperation of this method, as it was previously mentioned in section 5.1.

Agents method

This agents-based strategy is a decentralized cooperative method that was originally described on [122, 123]. The strategy makes use of solutions arranged in a matrix or "world" M. Each cell $M(i,j)$ contains a solution of the problem at hand, referred as $M(i,j).V$, and the cost of the solution $M(i,j).C$. There is no topological relation between the positions of the solutions in the matrix and their corresponding costs.

The second population is made by a set of k agents $\mathcal{A} = \{a_1, a_2, \ldots, a_k\}$ that move in the world M following certain rules. It is assumed that k is much lower than the number of solutions available. Each agent is rep-

Algorithm 18 Update Solutions Method pseudocode (minimisation)

```
procedure UpdateSolutions()
  for all a_i ∈ A do
    UpdateMemory(a_i.currentSol,a_i.cost)
    Update position two worst solutions in CIR (worst_1.(x,y),
    worst_2.(x,y))
    if a_i.currentSol is better than M(a_i.x,a_i.y).V then
      previousSolution ← M(a_i.x,a_i.y)
      M(a_i.x,a_i.y).V ← a_i.currentSol
      M(a_i.x,a_i.y).C ← a_i.cost
      Update best solution with a_i.currentSol, a_i.cost
      %improvementHistory ← (averageMem − a_i.currentSol)/averageMem
      %improvement ← (previousSolution − a_i.currentSol)/previousSolution
      if (%improvementHistory > %BigImprovement) AND
      (%improvement > %BigImprovement) then
        M(worst_1.x,worst_1.y) ← a_i.currentSol.
        sol ← generateGradientSol(previousSolution, a_i.currentSol).
        Update best solution with sol
        M(worst_2.x,worst_2.y) ← sol.
      end if
    end if
  end for
```

resented by a 3-tuple: $a_i = \{(x, y), currentSol, cost\}$ where (x, y) indicates the position in M where the agent lies and *currentSol* is the current solution being manipulated by a_i. The objective value of *currentSol* is stored in the variable *cost*. We will refer to each component of a particular agent a_i using the dot notation, so $a_i.x$ will stand for the X coordinate in the world where a_i lies.

On top of the basic strategy, we add a centralised information repository (CIR). It stores the cost and the position of the two worst solutions.

The global strategy is described in Algorithm 17. After a random initialisation of solutions and agents positions, several processes are applied. On *Move Agents*, each agent a_i moves to the best cell of the neighbourhood of the current position (eight cells around the position $a_i.(x, y)$). If there is no better cell in the current neighbourhood then it moves randomly to one of them.

Change Solutions is applied for every agent as follows: first step is to copy the cell solution into the agent: $a_i.currentSol = M(a_i.x, a_i.y).V$ and $a_i.cost = M(ai.x, ai.y).C$. Then, a_i generates a new random solution inside a circle of radius *PerturbationRadius* around $a_i.currentSol$.

Now, this new solution $a_i.currentSol$ may improve $M(a_i.x, a_i.y).V$ or not and actions should be taken for each option. These actions are encapsulated within the *Update Solutions* stage, which is fully described in Algorithm 18.

The procedure *UpdateMemory* sends the current agent's solution and its cost to a register Mem which is updated in a first-in first-out manner when it is full. The length of Mem is fixed on $MemSize$. The other initial steps are clear: the two registers in CIR for the two worst solutions are updated. If the new solution obtained by the agent is better than the one stored in the cell, then the cell's solution and its cost are updated. Afterwards, the algorithm checks whether the solution generated greatly improves the average of the costs stored in Mem and the previous solution in that position. If so, the 2 worst solutions on the "world" are replaced following the diagram shown on Figure 5.1. The figure shows the solution to which the agent performed the improvement (*previousSolution* that is discarded) together with the new generated solution (*currentSolution*, that is the one that has obtained a big improvement in the fitness, $a_i.currentSol$). $a_i.currentSol$ will replace *previousSolution* in the grid where the agent is and it will also overwrite the worse solution of the grid. Additionally, the second worse solution in the grid will be replaced with the solution *sol* that is generated in function of the euclidian distance (*distance*) that separated *previousSolution* and $a_i.currentSol$. *sol* is made in the function *generateGradientSol()* by generating in first place an intermediate solution (the smallest circle) that is at three times the distance from *previousSolution* that $a_i.currentSol$ was. Then, a perturbation of radius *distance* is applied over this intermediate solution to obtain the solution *sol* inside the biggest circle of the figure.

The method receives the following parameters (the values they are set to are also given):

- $rows = 5$: Number of rows in M.
- $columns = 10$: Number of columns in M.
- $Agents = 4$: Number of agents.
- $PerturbationRadius = 14.0$: Radius for the perturbation of solutions.
- $MemSize = 140$: Size of the Mem.

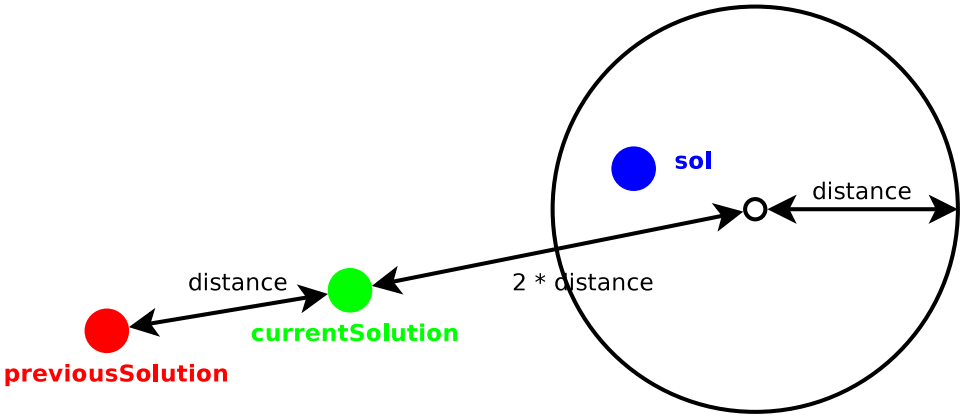


Figure 5.1: When the improvement done by the agent from *previousSolution* to $a_i.currentSol$ (*currentSolution* in the picture) is big, the worst solution in the grid is also replaced with $a_i.currentSol$. Furthermore, the solution *sol* is generated inside the biggest circle to replace the second worst solution of the grid.

- $BigImprovement = 60\%$: Percentage of improvement to be considered as big.

The Agents algorithm can bee seen as a population-based decentralized cooperation algorithm since the improvements that one agent does in M can be seen by the other agents. When an agent is moving to the best neighbor solution of the "world" it will get attracted by good grid cell solutions that were stored by other agents. Therefore, as the agents move through the grid they will try to perform further improvements on the cells that other agents have already improved.

5.3.3 Further details of the cooperative strategy

To finish with this section, we are going to give the details of the implementation done to deal with DOP's. In first place, we simulate the paralellism in the same way than in chapter 4. The parameter $freq\_interval$ was set to the interval [50, 60] basing on previous experimentation.

The cooperative strategy is composed by 12 solvers which implement the method described in section 5.2.3 and they differ in terms of the length of the initial step size, δ_{init}, (three different values for this parameter are considered) as well as the initial solution. The parameter setting for the tabu search is displayed in Table 5.2. σ is defined as the maximum of the

Parameter	Description	Value
δ_{init}	initial length of the operator step	$\sigma/2, \sigma/3, \sigma/5$
resolution	minimal step value	0.001
LRR	radius for the local minimum regions	$0.01 \cdot \sigma$
maxNoImp	maximum number of non improvement movements	3
tenure	length of the tabu list of movements	1

Table 5.2: Parameter setting for the tabu search. The different values set to the parameter δ_{init} for each solver are also displayed.

difference between the lower and the upper limit of each variable.

Regarding the parameters of the coordinator, the radius for the visited regions ρ was set to $0.15 \cdot \sigma$ and the antecedent threshold $\lambda_{reaction}$ to 1. The mutation operator used to modify the best global solution consists on selecting randomly a direction following a uniform distribution and then, take the point at a distance r from the best global solution. In this case r is set to $0.1 \cdot \sigma$.

5.4 Results

Experiments have been conducted over MPB and the three real test functions changing the position of the optimal values (peaks) with 5 evenly distributed percentages of change severity between a 2% and a 10% of the full range of possible coordinate values for each problem. The rest of the parameters in the case of MPB are taken from the Scenario 2 defined by Branke in [22] and previously presented on Table 5.1. The real test functions are configured to use 5 dimensions as in the case of MPB and 1, 3, 5 or 10 functions of the same type. Changes to the function's optimum were performed every 5000 function evaluations, and a run grouped 100 consecutive changes. Each experiment consisted in 30 independent runs, each of them with its own random seed.

One of the most currently accepted measures for dynamic optimization is the *offline error*, proposed in [22], which is the running average of the difference between the optimum values and the best solution encountered so far, at any time between two consecutive changes. When a change occurs,

Severity	Agents	multi-QPSO	CCS
2	4.610(2.683)	6.702(4.889)	3.576(2.462)[*]
4	4.789(2.701)	7.011(4.882)	4.042(2.469)[*]
6	5.040(2.819)	6.843(4.465)	4.477(2.515)[*]
8	5.270(2.821)	6.541(4.015)	4.811(2.582)[*]
10	5.445(2.911)	6.582(4.039)	5.035(2.580)[*]

Table 5.3: MFE results for the MPB problem

the offline error is reset. This measure is always greater than or equal to zero and is defined as:

$$offline\ error = \frac{1}{T}\sum_{t=1}^{T}(optimum - bestSolution) \tag{5.16}$$

In this way, the offline error gives a good idea of the average distance to the optimum of the solutions given by the algorithms throughout their execution. The lower the value of its offline error, the better an algorithm behaviour is considered.

For the experiments conducted, the offline error of each algorithm just before a change was recorded, and this value was averaged over all the 100 changes of a run, for all the 30 independent runs. This procedure leads to the Mean Fitness Error (MFE) defined by Richter and Yang [132] as:

$$MFE = \frac{1}{R}\sum_{r=1}^{R}\left[\frac{1}{T}\sum_{t=1}^{T}\left(f(x_s(k),k) - \max_{x_j(t)\in P(t)} f(x_j(t),k)\right)\right]_{k=\lfloor\gamma^{-1}(t)\rfloor} \tag{5.17}$$

where $\gamma \in \mathbb{N}$ is the change frequency, $\max_{x_j(t)\in P(t)} f(x_j(t), \lfloor\gamma^{-1}t\rfloor)$ is the fitness value of the best-in-generation individual $x_j(t) \in P(t)$ at generation t, $f(x_s(\lfloor\gamma^{-1}(t)\rfloor), \lfloor\gamma^{-1}(t)\rfloor)$ is the maximum fitness value at generation t, T is the total number of generations used in each run, and R is the number of consecutive runs. In our case, since the algorithms proposed do not use a clear concept of generation we will consider that a generation corresponds to the period between two consecutive fitness function changes.

The results are presented in Tables 5.3, 5.4, 5.5 and 5.6. These tables

Table 5.4: MFE results for the Ackley function

Functions	Severity	Agents	multi-QPSO	CCS
1	2	2.227(0.891)	2.219(1.084)	0.598(0.529)[*]
1	4	2.340(0.868)	2.778(1.303)	0.886(0.511)[*]
1	6	2.462(0.962)	2.933(1.382)	1.118(0.547)[*]
1	8	2.552(0.857)	3.017(1.463)	1.349(0.573)[*]
1	10	2.659(0.775)	3.074(1.462)	1.512(0.570)[*]
3	2	2.246(0.664)	1.711(0.755)	0.563(0.431)[*]
3	4	2.343(0.528)	2.108(0.835)	0.889(0.459)[*]
3	6	2.460(0.488)	2.202(0.837)	1.137(0.471)[*]
3	8	2.573(0.509)	2.248(0.786)	1.347(0.481)[*]
3	10	2.677(0.487)	2.312(0.756)	1.524(0.508)[*]
5	2	2.270(0.476)	1.488(0.633)	0.536(0.394)[*]
5	4	2.395(0.501)	1.818(0.699)	0.854(0.408)[*]
5	6	2.488(0.441)	1.886(0.661)	1.095(0.423)[*]
5	8	2.572(0.454)	1.916(0.600)	1.289(0.434)[*]
5	10	2.696(0.489)	1.977(0.576)	1.490(0.439)[*]
10	2	2.249(0.412)	1.299(0.623)	0.402(0.326)[*]
10	4	2.385(0.398)	1.491(0.646)	0.694(0.335)[*]
10	6	2.456(0.392)	1.598(0.595)	0.931(0.340)[*]
10	8	2.553(0.372)	1.685(0.545)	1.131(0.347)[*]
10	10	2.667(0.378)	1.693(0.487)	1.332(0.366)[*]

show the final MFE of the 30 independent runs, with the standard deviation in parenthesis. Here, values with an asterisk ([*]) indicate that the corresponding algorithm obtained the best (lowest) MFE. For every configuration of the problem, the Kruskal-Wallis non-parametric test for multiple comparisons has been used to asses the diferences between the performances of the three methods. The null hypothesis could not be rejected at significance level 0.01 for all problem configurations. Apart from this, the Wilcoxon's unpaired rank sum test at a confidence level 0.05 was performed between the best algorithm and each of the others, to assess if there were statistical differences. If there was no difference, the other algorithm's value is shown with the word "[NO]".

The MPB results on Table 5.3 show that multi-QPSO obtains very good

MFE results, but both of the other methods with some kind of explicit cooperation (Agents and the centralised cooperative strategy (CCS)) outperform multi-QPSO in the MPB. Moreover, CCS is significantly better than the other two methods while the Agents algorithm follows closely, being better than multi-QPSO with a significant difference in performance. It is also clear that as the severity increases the MFE gets worse (higher) with the only exception of multi-QPSO where the MFE shows no correlation with the severity. This is probably because while the other methods are able to improve their MPB results when there is a lower severity, multi-QPSO is probably unable to quickly keep track of the peak movements even with the lowest tested severity.

For the Ackley real test function (Table 5.4), CCS shows far better results than the other two methods with a difference that is again statistically relevant. Multi-QPSO and Agents shows mixed results with Agents outperforming multi-QPSO for most of the experiments with only one function, while multi-QPSO becomes better as the number of functions increases. It can also be seen that the severity affects all methods similarly but in a smaller amount than in the MPB case, with only slighty worse results for every method as the severity increases.

With the Griewank function (Table 5.5), all methods are able to obtain a very low MFE but again CCS is better in almost all cases with just one case (3 functions and a severity of 2%) where it is outperformed by multi-QPSO and another one where there is no statistical difference with the multi-QPSO result (5 functions and a severity of 2%). The severity also affects the algorithms in a very similar way than in the ackley function with slightly worse results as the severity increases.

Finally, Rastrigin (Table 5.6), the most difficult function, shows the biggest difference between the methods. In this function, Agents is clearly worse than the other two methods and CCS is again by far the best of the three methods. In conclusion, all the tables have shown that CCS is almost allways the algorithm with the lowest MFE in a statistically significative manner. The only two exceptions for this statement are the two specific cases of the Griewank function that were pointed out in the previous paragraph. Being this a more difficult to optimize function, the severity also affects the methods more than in the previous tests, because after a fitness

Table 5.5: MFE results for the Griewank function

Functions	Severity	Agents	multi-QPSO	CCS
1	2	0.132(0.074)	0.099(0.042)	0.089(0.039)[*]
1	4	0.180(0.067)	0.156(0.057)	0.107(0.038)[*]
1	6	0.220(0.077)	0.202(0.087)	0.107(0.039)[*]
1	8	0.251(0.084)	0.235(0.103)	0.114(0.040)[*]
1	10	0.260(0.088)	0.268(0.127)	0.116(0.042)[*]
3	2	0.178(0.063)	0.094(0.036)[*]	0.101(0.0439)
3	4	0.226(0.064)	0.158(0.055)	0.119(0.046)[*]
3	6	0.252(0.070)	0.204(0.082)	0.124(0.049)[*]
3	8	0.265(0.074)	0.249(0.132)	0.127(0.046)[*]
3	10	0.278(0.079)	0.282(0.144)	0.131(0.048)[*]
5	2	0.161(0.069)	0.125(0.063)[NO]	0.119(0.049)[*]
5	4	0.220(0.064)	0.166(0.062)	0.134(0.048)[*]
5	6	0.254(0.071)	0.229(0.091)	0.135(0.051)[*]
5	8	0.270(0.076)	0.255(0.121)	0.142(0.052)[*]
5	10	0.282(0.079)	0.295(0.123)	0.147(0.053)[*]
10	2	0.182(0.061)	0.125(0.062)	0.122(0.043)[*]
10	4	0.241(0.061)	0.192(0.061)	0.141(0.043)[*]
10	6	0.252(0.063)	0.239(0.078)	0.143(0.044)[*]
10	8	0.264(0.064)	0.263(0.084)	0.148(0.047)[*]
10	10	0.275(0.065)	0.293(0.096)	0.153(0.046)[*]

function change occurs it is also more difficult to reposition the solution near the good search space locations.

While the results in terms of the best values obtained for each fitness function change are very important, it is also important to know how the methods perform during the whole search process and not just before each fitness function change.

To analyze this behaviour, a graph of the best results of all the methods against the optimum for a sample run of each test function is generated. In order to compair the methods in the worst scenario, the instances selected are the most difficult ones where the number of functions is maximum and the biggest severity (10%) of change is used. For each selected run, the first 20000 evaluations were plotted taking values every 100 evaluations. Since

Table 5.6: MFE results for the Rastrigin function

Functions	Severity	Agents	multi-QPSO	CCS
1	2	10.503(3.019)	2.288(1.300)	0.882(0.861)[*]
1	4	13.189(3.091)	4.754(1.673)	1.224(0.962)[*]
1	6	13.895(3.160)	6.880(2.323)	1.745(1.046)[*]
1	8	14.245(3.226)	7.686(2.615)	2.228(1.188)[*]
1	10	14.473(3.212)	8.229(2.789)	2.496(1.273)[*]
3	2	10.814(2.883)	1.849(0.695)	1.094(0.802)[*]
3	4	13.121(2.849)	4.568(1.411)	1.397(0.866)[*]
3	6	14.058(2.915)	6.903(2.089)	1.925(0.950)[*]
3	8	14.327(2.943)	7.693(2.363)	2.353(1.032)[*]
3	10	14.358(2.965)	8.000(2.435)	2.536(1.086)[*]
5	2	10.525(2.759)	1.786(0.791)	1.041(0.722)[*]
5	4	13.076(2.763)	4.562(1.420)	1.334(0.761)[*]
5	6	13.969(2.764)	6.840(2.108)	1.854(0.828)[*]
5	8	14.078(2.807)	7.678(2.298)	2.294(0.971)[*]
5	10	14.161(2.817)	7.865(2.354)	2.473(1.003)[*]
10	2	10.189(2.585)	1.939(0.722)	0.826(0.601)[*]
10	4	12.473(2.597)	4.465(1.393)	1.133(0.672)[*]
10	6	13.175(2.626)	6.511(2.002)	1.667(0.802)[*]
10	8	13.310(2.615)	7.132(2.156)	2.054(0.872)[*]
10	10	13.379(2.640)	7.355(2.251)	2.194(0.913)[*]

the fitness function changes every 5000 evaluations, this 20000 evaluations correspond to the evolution of the fitness for the first four fitness function changes.

The results for MPB are shown on Figure 5.2. Since this is a maximization problem, the best fitness obtained by the methods is always reduced when a fitness function change occurs and then starts growing again as the methods make progresses during the search. The first thing to notice in the figure is that CCS never the best method at the first evaluations after a fitness function change. Despite that, CCS is able to continue improving the solutions at a higher pace for a much longer number of evaluations than the

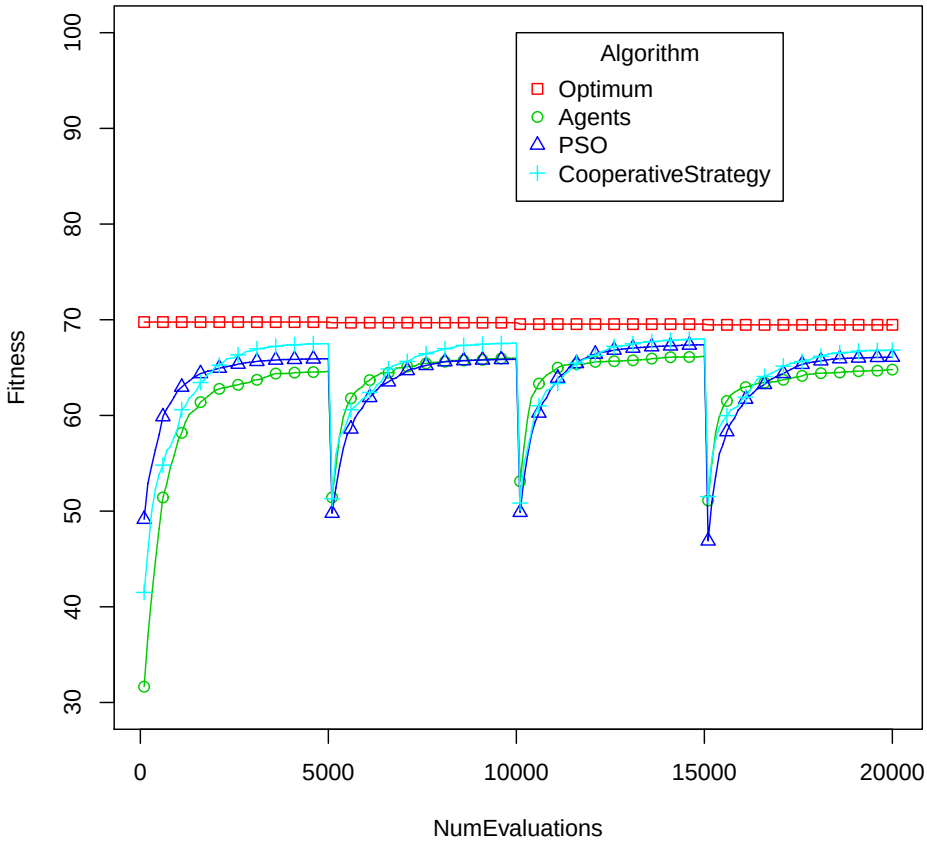


Figure 5.2: Best algorithm results vs optimum for the MPB problem with a 10% of severity

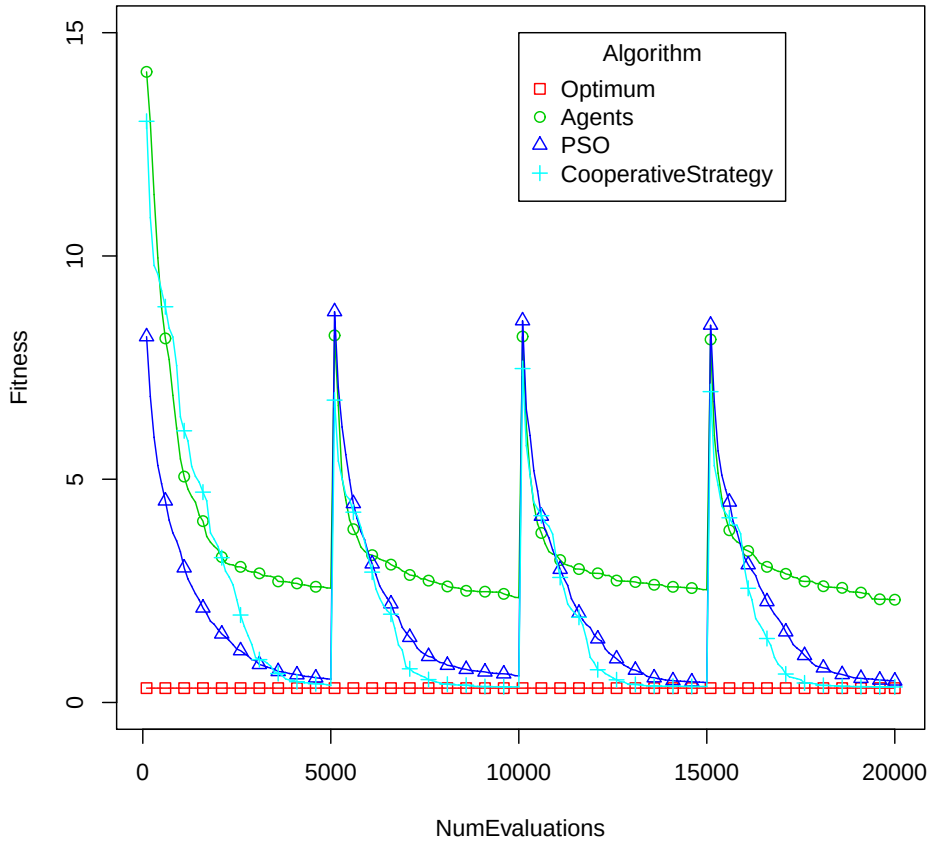


Figure 5.3: Best algorithm results vs optimum for the Ackley function with 10 functions and a 10% of severity

other methods leading to the statistically better results we have seen before when considering the moments just before a fitness function change. Therefore, CCS is clearly better than the other methods only when the number of fitness evaluations without fitness function changes is high enough, but for fewer evaluations multi-QPSO and specially Agents are clearly better. In three out of four of the function changes seen on the figure, Agents is the best algorithm for the first 1000 evaluations after a fitness function change.

In the real test functions case, as they are minimization problems, the fitness after a function change starts with a high value and it goes down as the methods improve their solutions. The results for Ackley are presented on Figure 5.3. In this case all three methods perform very similarly in the first evaluations after a change. It is when more evaluations are available when the differences among the methods appear. Agents seems to get stuck more quickly with very little improvement on the second half of the search stage for each change. On the contrary, multi-QPSO and CCS keep improving on this second half of evaluations with the improvements of CCS showing a particular good slope. In this way, CCS almost matches the optimal values in the last evaluations of each function change with multi-QPSO following closer and Agents at a big distance from the other methods. It is clear that the intensification of the search done by CCS is much better than the one of the other two methods. This suggest that the combination of the centralized cooperation with the trajectory solvers gets a very good symbiosis. The trajectory solvers are particularly well suited for intensification of the search and the cooperation helps to avoid local optima which could explain the increase of the slope in the improvement of fitness at several points of the search process.

Figure 5.4 contains the results for the Griewank function where Agents and multi-QPSO show a very similar behavior during the whole evolution of the search. CCS again improves both of the other methods on the results at the latter stages of each fitness function change. But this time, CCS is also better at the first evaluations after a fitness function change because it is able to regain good results with fewer evaluations.

Finally, Figure 5.5 shows the corresponding plot for the Rastrigin function. On the contrary to the previous cases, the performance order of the three algorithms keeps almost the same for the whole evaluations with CCS

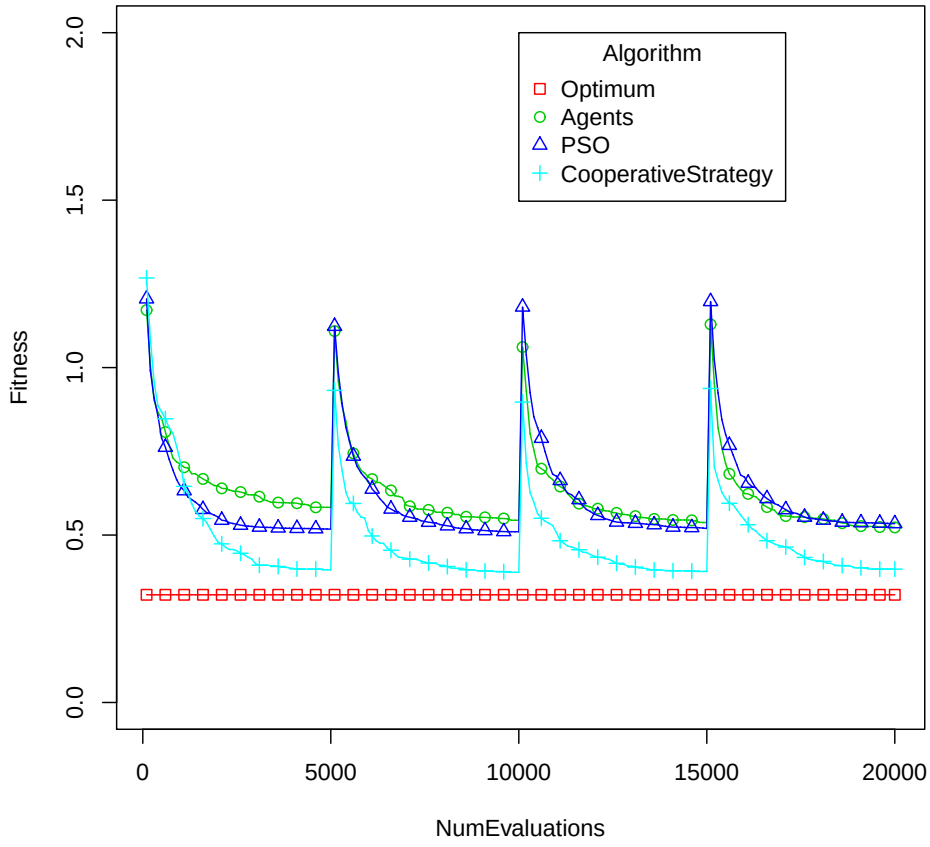


Figure 5.4: Best algorithm results vs optimum for the Griewank function with 10 functions and a 10% of severity

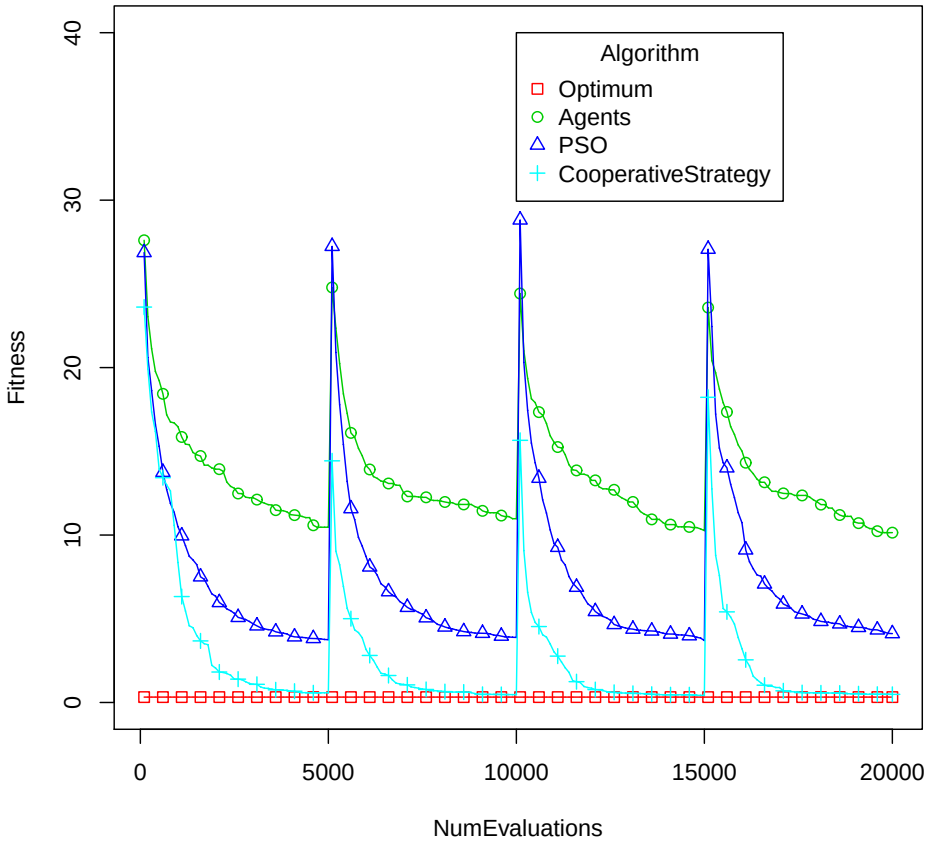


Figure 5.5: Best algorithm results vs optimum for the Rastrigin function with 10 functions and a 10% of severity

greatly outperforming the other methods and Agents as the worse algorithm. Despite the fact that this function is considered to be difficult to optimize, CCS is able to get near optimal values in a consistent way. This can be explained by two factors. Firstly, the tabu search implemented by the solvers of the CCS is probably well suited to optimize this kind of functions. This suggest that the believe that trajectory based methods are not good for dynamic optimization should be reconsidered. At the very least, when these trajectory based methods are combined with a cooperation scheme, the results shown here demonstrate that it is possible to obtain a good sinergy that can both improve the optimization results and avoid getting stuck in local minima.

5.5 Conclusions

In this chapter we have tackled the study of a new scenario to apply centralised cooperative methods. We have chosen Dynamic Optimisation Problems (DOPs) and concretely, those ones whose objective function change over the time. Two main reasons lead us to this election: a) there is a growing interest on the resolution of these problems due to its closeness to real-world situations (trade market predictions, meteorological forecast, etc) and b) as far as our knowledge is concerned centralised cooperative strategies have not been applied to these problems before. The strategy developed to deal with these problems had the same structure than the one used in chapter 4. The solvers implemented a trajectory-based algorithm (tabu search) with different parameter configurations. We saw that this type of methods can be easily adapted to DOPs by adding a mechanism that controls the objective function changes and programming the restart of certain memories every time that one of this changes takes place.

The performance of the centralised cooperative strategy has been compared with two other methods of the literature: multi-QPSO and Agents. The multi-QPSO method was chosen as a reference, because it is a well known method with good performance. In terms of cooperation, multi-QPSO can be seen as a method with an implicit cooperation between the different particles of each swarm. Agents was chosen because it also uses an explicit cooperation scheme but it is also a population-based method that does not use trajectory solvers.

Agents used a decentralized cooperation scheme on opposition to the centralized cooperation that was present on CCS. Despite that, Agents has shown to be able to outperform multi-QPSO in the MPB and some of the Ackley real test function experiments, while its results were worse than multi-QPSO in the remaining Ackley configurations and the other two real test functions. This suggests that the cooperation included in Agents provides some benefits over multi-QPSO on the easier problems, but since the optimization done in Agents relies only on using simple random perturbations of solutions it may not be enough to cope with the most difficult problems even with the help of the cooperation.

On the other hand, the centralised cooperative strategy used an explicit centralized cooperation scheme together with complex trajectory-based optimizers (tabu search). The method showed very promising results that are able to outperform Agents and multi-QPSO in most of the test problems. It was also very consistent with very good results for all test problems and configurations. Therefore, this kind of cooperation deserves a bigger attention in future works, in order to further assess their performance on different problems and to test different alternatives in terms of the cooperation policy as it has been done on the static case.

Moreover, it has been believed and assumed that population based methods were best suited to deal with DOPs because it was supposed that a bigger number of solutions could keep better track of a changing environment. While this still may hold true, the success of this centralised cooperative strategy suggest that trajectory methods combined with a cooperation scheme may have an important role to play in the dynamic optimization field and further research should be put into this area. Furthermore, this strategy can also be seen as a population algorithm if we take into account that each trajectory solver (tabu) has one solution, despite that the number of solutions is very low in comparison with the usual values for typical population-based algorithms. This also suggest that the sizes of the populations are of relative importance and while some methods may get better scores when using more solutions, others could benefit of a bigger intensification on a small number of solutions if they are able to avoid local optimum with a low number of solvers/solutions. This is also an open research area where there is a need to find a balance of population that can fit the needs of both the method and the problem and hand. At the end, this balance is

probably no more than the balance between intensification and diversification that is common to most optimization tasks but also coupled with the specific method needs and characteristics.

Chapter 6

A centralised cooperative strategy for solving multiple instances

In this chapter we present a cooperative strategy designed to solve *a set of instances*. The method is based on a set of operators and a basic learning process that is fed up with the information obtained while solving several instances. The output of the learning process is an adjustment of the operators. The instances can be managed sequentially or simultaneously by the strategy, thus varying the information available for the learning process. The method has been tested on different SAT instance classes and the results confirm that a) the usefulness of the learning process, b) that embedding problem specific algorithms into our strategy, instances can be solved faster than applying these algorithms instance by instance, and c) the simultaneous resolution shows a more robust behaviour than the simultaneous one.

6.1 Motivation

As we pointed out in the introduction of this dissertation, metaheuristics present some drawback when they come into practise whose impact can be attenuated by Hybrid Metaheuristics and cooperative strategies. Apart from these methods, in the literature we can find other proposals to alleviate such problems. One of them is self-adaptation of parameters that aims to reduce the human intervention in the practical application of metaheuristics

by means of the automatic tuning of critical parameters or operators during the search process to better solve the corresponding instance.

Many studies in these fields are oriented to solve one instance at a time, independently of the other ones. But, what if we need to solve *a set of instances*? For example, it is not hard to imagine a server that receives requests to solve instances of the same problem. Under this situation, which strategy should be used to efficiently solve all the instances?. A naive mode of operation would be to choose a particular algorithm and to solve the instances one by one. However, working in this way, valuable information obtained during and after the resolution process of every instance is discarded. Besides, the drawbacks we pointed out before are exacerbated. For example, it could be necessary to solve instances of different characteristics, thus the performance of the selected algorithm can be excellent for some instances but unacceptably bad for others. Of course the instances could be solved independently, but in parallel if time is a constraint.

To the best of our knowledge, there is a lack of such strategies in the literature despite their potential interest.

In [59], Gagliolo et al. face the resolution of a set of instances using a set of ad-hoc algorithms. In this approach the instances are solved one by one and a feedback mechanism is used to modify the amount of resources assigned to every algorithm. This modification is essentially based on performance information.

The aim of this chapter is to further explore strategies that can efficiently manage the solution of a set of instances. With this aim, we present a self-adaptive strategy, that can manage the instances one by one or all of them simultaneously. A learning mechanism promotes the use of those operators that shows good performance and it is important to remark that, depending on how the instances are processed, the information available for learning and adaptation is low (when one instance at a time is processed) or high (all instances are being solved in parallel). As a test bed, we use the SAT problem and SAT-specific operators are employed. The strategy's behaviour is evaluated over a wide variety of instance types and we will provide comparisons against some SAT-specific algorithms.

The rest of the chapter is organised as follows. We will start giving some background about self-adaptation strategies in 6.2. Section 6.3 describes the proposed strategy and the adaptation mechanism. The details of the test

bed and the different classes of instances used, the SAT specific operators and the implementation details are provided in Section 6.4. Then, in Section 6.5 we present the computational experiments performed to test the benefits of our strategy and the results obtained as well as an analysis of its dynamic behavior. Finally, Section 6.6 is devoted to conclusions.

6.2 Self-adaptation strategies

The research in this area is addressed to design mechanisms that tune critical parameters during the search process in order to fit them to the corresponding instance or stage of the search. This topic has been widely explored in some fields such as evolutionary algorithms [95] or local search [9]. In the case of evolutionary algorithms, some methods have been proposed to adapt the mutation rate [7, 94], crossover probabilities [152] or both of them [153], exploration operator probabilities [140], the local search operator ('memes') in memetic algorithms [75, 117, 138], etc. Reactive search [9], previously described in 4.2.2, could be considered as an adaptation strategy for local search algorithms. As we saw, its objective is to give a ready response to events during the search through an internal online feedback loop for the self-tuning of critical parameters. These ideas have been applied to some local search methods such as Variable Neighborhood Search [23], Simulated Annealing [53], or Tabu Search [11].

Other types of self-adaptation strategies can be found in some "Algorithm Portfolios". Here, a set of algorithms are run in parallel until the fastest one solves the problem. The amount of resources available for each one is given as a function of its performance. This resource assignment can be done by modeling the performance of the candidate algorithms during a certain period of time [81], by means of reinforcement learning [90], dynamic programming [126] or bandit problem solvers [59] among others.

6.3 Scheme of the cooperative strategy for solving multiple instances

In simple terms, our strategy can be seen as a method that co-evolves solutions from different instances, jointly with a set of operators. Since many

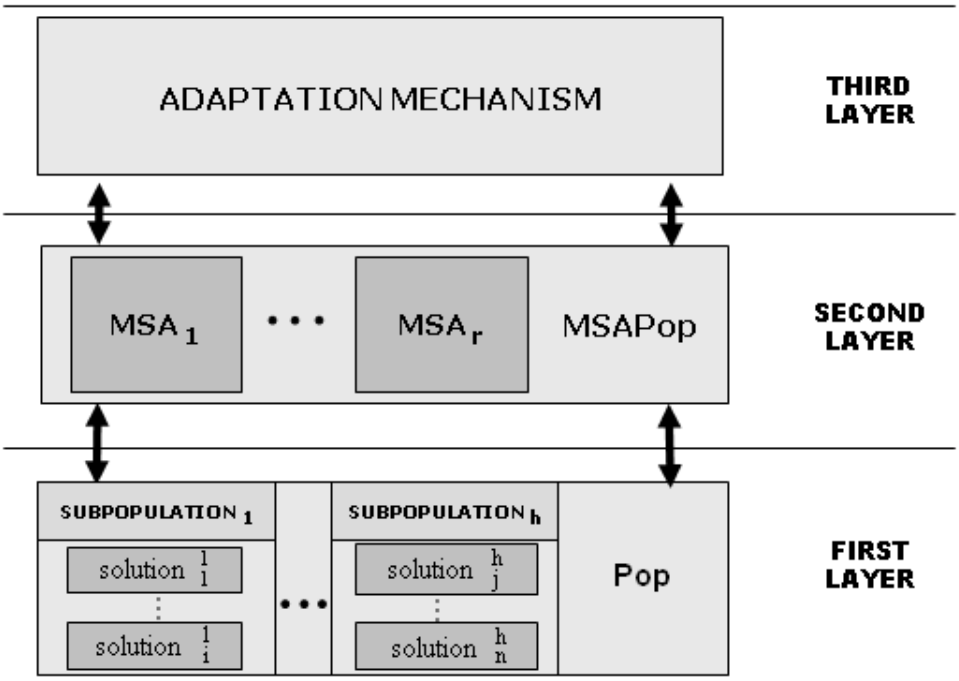


Figure 6.1: Scheme of the strategy

instances are going to be solved, we need to define an *operational mode*: instances can be processed simultaneously or sequentially. Since many instances are available, it is possible to reinforce or adapt the strategy in order to promote those operators that are giving the best results. This *learning stage* can be switched on or off.

The proposed strategy is depicted in Figure 6.1, where three layers are shown.

Let us assume that we have a set of instances I from a particular optimisation problem to be solved. We define f_j as the objective function of such problem, where $j \in I$.

The first layer of the strategy is composed by a population of solutions $Pop = \{s_1^1, \ldots, s_n^h\}$, where s_i^j indicates that the solution i belongs to the instance $j \in I$. A sub-population Pop_j is defined as $Pop_j = \{s_i^l \in Pop | l = j\}$. When necessary, we may omit the instance index for the sake of simplicity.

The second layer is composed by a set of Modification Solution Agents (MSA) that cooperate to improve the solutions in Pop using certain operators. The set of $MSAs$ available constitutes $MSAPop$.

Every $MSA_i \in MSAPop$ is defined by a tuple:

$$MSA(O_k, G_{accept}, credit) \tag{6.1}$$

where O_k is an operator that belongs to the set of operators $\Theta = \{O_1, \ldots, O_m\}$ (they can be interpreted as mutation or move operators), $credit \in \Re$ stores the amount of "credit" obtained by the agent and G_{accept} is an acceptance criterion that can be stochastic (as in simulated annealing), tabu, fuzzy [121], etc.

The third layer is where the learning and adaptation mechanism takes places: its responsibility is to adapt the operators that every MSA's uses.

As we stated before, two main characteristics define our strategy: the operational mode and the learning stage. Both will be described next.

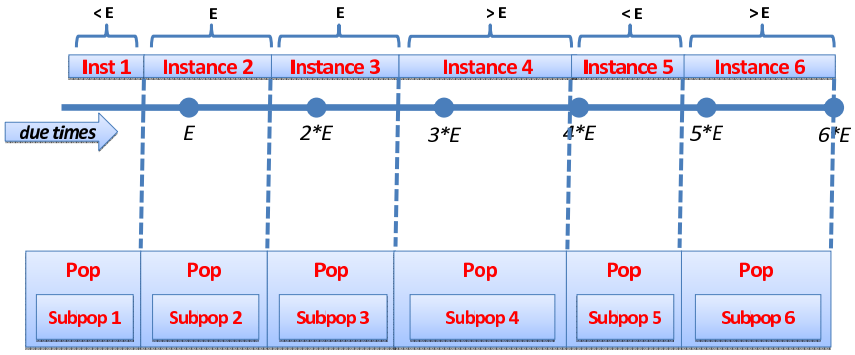
6.3.1 Operational Mode

The *set of instances* that needs to be solved can be processed simultaneously or sequentially. Figure 6.2 shows both operational modes when 6 different instances are used. When instances are solved sequentially, each one is allocated a total of E evaluations, and we ensure that instance i processing should finish when $E \times i$ evaluations have been done. In other words, we fix a due time for every instance. In the parallel or simultaneous mode, $6 \times E$ evaluations are allocated to solve all instances. Every instance has, at least, E evaluations to be solved. When an instance is solved, its corresponding subpopulation is removed from Pop. In this way, the strategy works over those instances that have not yet been solved.

Figure 6.3 shows an example of the operation of the second layer using four MSA and twelve solutions. The set of MSA is moved over Pop as a window, modifying slots of four solutions at each iteration. A particular MSA_i receives as input a solution s, applies its operator O_k and obtains a new solution $s\prime$. Then, if $G_{accept}(s, s\prime) = True$ then the new solution replaces the older one and MSA_i is given certain credit $\Delta credit(s, s\prime)$. If $G_{accept}(s, s\prime) = False$, then the current solution is kept.

In the example, three iterations are enough to cover all the solutions from all the instances. When this occurs, the process is repeated. The credit assignment process used here will be described later, but the reader

a) **Sequential mode**



b) **Parallel mode**

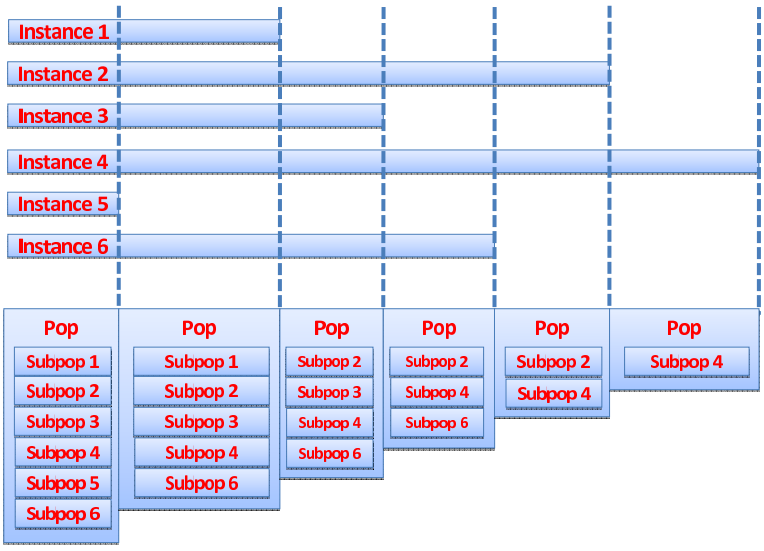


Figure 6.2: Examples of both operational modes over 6 instances.

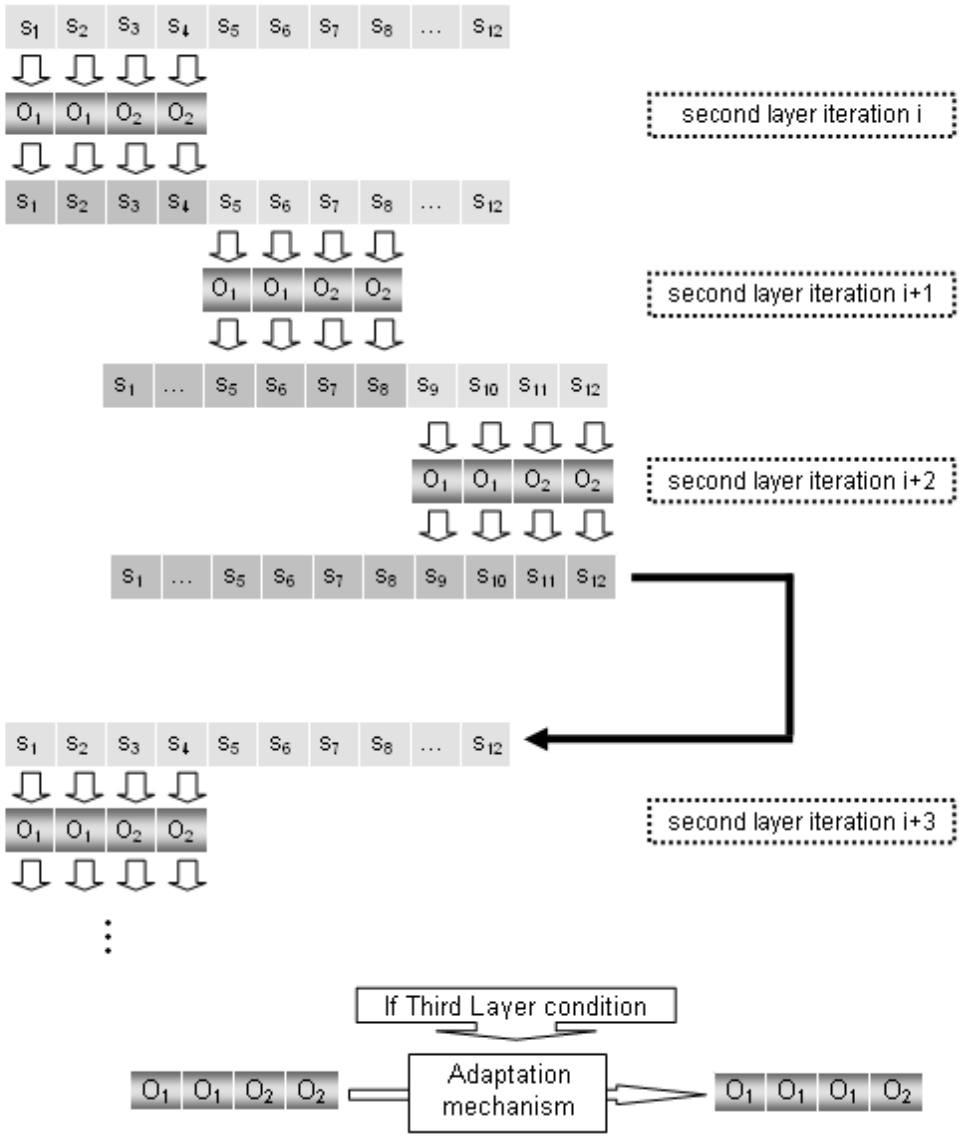


Figure 6.3: Example of the working of the strategy with 4 MSA, 12 solutions and 2 operators.

should be aware that many alternatives are possible. See [139] for an up to date review.

A crucial difference between both operational modes is what happens when an instance j was solved using $E\prime < E$ evaluations. When the sequential mode is applied, the remaining $E - E\prime$ evaluations are automatically assigned to the next instance $j+1$ that, as a consequence would have $B_{j+1} = E + (E - E\prime)$ evaluations available (recall that just the due time should be enforced). Then, if instance $j+1$ is solved in $E\prime\prime < B_{j+1}$, the $B_{j+1} - E\prime\prime$ non used evaluations are assigned to instance $j+2$ and so on. In the case that one of these instances is "hard", then all the potential gain in time is consumed in just one instance. When the simultaneous mode is used, the concept of evaluations assigned to instances does not exist. While the strategy have evaluations available, then it will simultaneously operate over all the instances.

6.3.2 Learning Stage

The learning stage can be used or not with every operational mode. Two elements need to be defined in this stage: how the adaptation takes place and when it should be applied.

How to adapt

When the adaptation mechanism is triggered by certain condition, the third layer assigns operators to the MSA_i using the credit information to decide. The learning and adaptation process is based on a vector of weights w, where the component w_k represents a performance measure associated with operator $O_k \in \Theta$ ($w_k \in \Re$, $length(w) = |\Theta|$).

At the i^{th} iteration of this layer, the adjustment procedure follows the next three stages:

1. *Weights update*: for each operator $O_k \in \Theta$:

   (a) A reward $r_k(i)$ is calculated as the total amount of "credit" gained, since the last update, for those $MSA \in MSAPop$ that had O_k as operator .

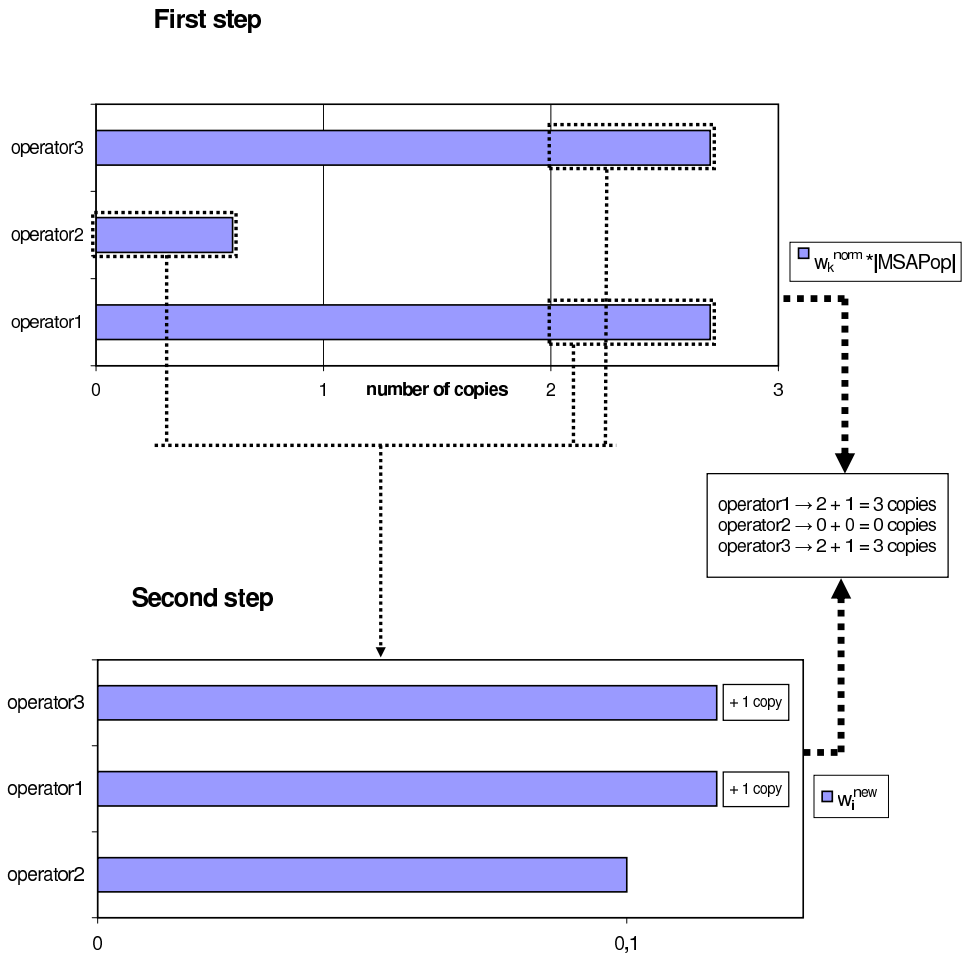


Figure 6.4: Graphical example of one application of the adaptation mechanism with 3 operators and 6 MSA

(b) The new weight at iteration i, $w_k(i)$, is calculated as:

$$w_k(i) \leftarrow w_k(i-1) + r_k(i) \tag{6.2}$$

(c) The weight is normalised ($m = |\Theta|$):

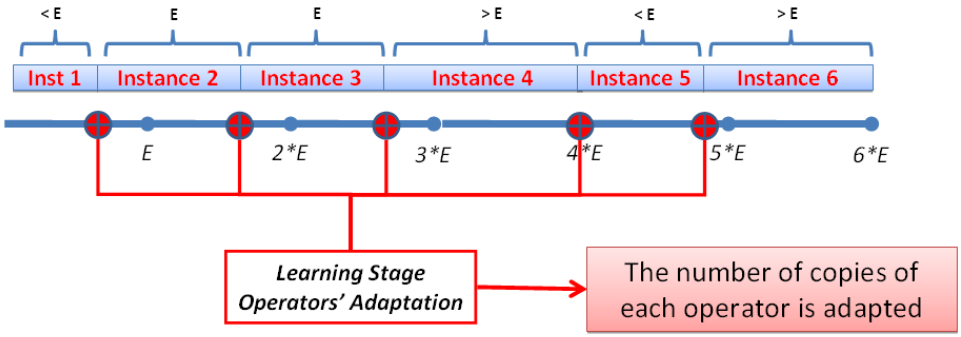
$$w_k^{norm} = \frac{w_k}{\sum_{i=1}^{m} w_i} \tag{6.3}$$

2. *Assignment of number of operator copies*: this stage is divided in two steps. An example of the process is shown in Figure 6.4:

(a) An initial number of copies is determined as:

$$copies_k^1 = [w_k^{norm} \times |MSAPop|] \tag{6.4}$$

a) **Sequential mode**



b) **Parallel mode**

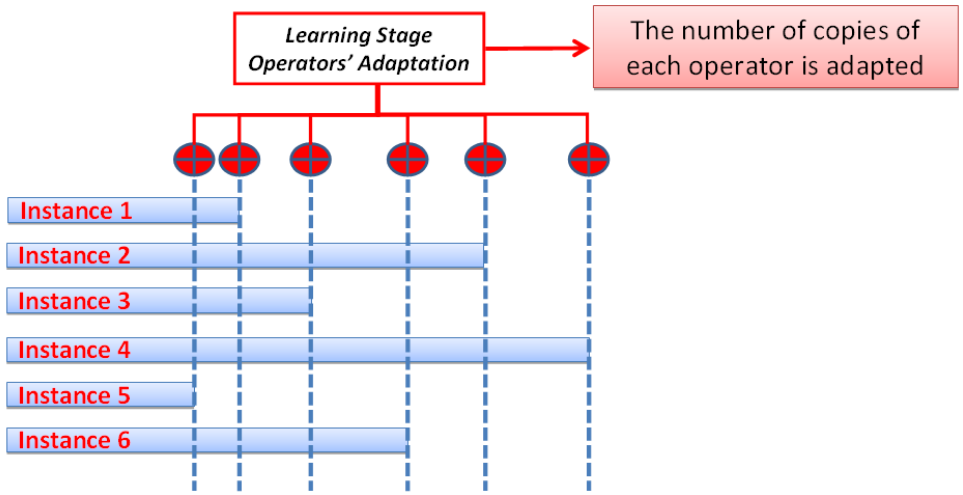


Figure 6.5: Schemes of both operational modes with the learning stage activated. Each time an instance is solved, adaptation takes place. In a) no modifications are made while an instance is being solved. In b) the adaptation may affect the ongoing solving processes.

where [] stands for the integer part. As we can see in Figure 6.4, in this step the mechanism determines the number of fragments of size $\frac{1}{|MSAPop|}$ that corresponds to each normalised weight.

(b) Additional copies calculation: For each $O_k \in \Theta$, a new weight is calculated

$$w_k^{new} = w_k^{norm} - \frac{copies_k^1}{|MSAPop|} \tag{6.5}$$

The operators are sorted in decreasing order of w_k^{new} and then, from top to bottom in this list, the remaining number of copies are then assigned, one by one, to these operator. This value is called $copies_k^2$ and random selection is used when a tie occurs.

3. *MSA adaptation.* Every operator O_k is randomly assigned to as many MSA as indicated by the $copies_k^1 + copies_k^2$ value.

When to adapt

The answer to this question is simple: trigger the adaptation when an instance is solved. Figure 6.5 shows a scheme highlighting the time steps where adaptation is triggered. Given the adaptation scheme described before, again we can observe crucial differences between both operational modes. In the sequential case Fig. 6.5 (a) we can observe that the operator that solved the first instance, have a higher chance of getting more copies. However, this "learning" is just available to the second instance in the list. Moreover, the definition of the MSA is kept fixed during the whole resolution of the instance. In turn, when the simultaneous mode is used, the operators in the MSA can be changed during the resolution. In this way, those non successful operators can be literally filtered out from the resolution process.

6.3.3 Classifying the strategy

In this subsection, the categorization of the centralised cooperative strategy were done following the same lines as in the former chapters. Such categorization is the following:

- **Talbi's taxonomy**:
  - *hierarchical taxonomy*: low-level teamwork hybrid (the MSA are not self-contained methods)

  - *flat taxonomy*: general and global method with heterogeneous nature (operators are different).

- **Raidl's taxonomy**:
  - *what is hybridized*: this strategy has also two hybridization types. In the, metaheuristics are combined with metaheuristics and these ones, for their part, are hybridized with a self-adaptation mechanism.
  - *level of hybridization*: low-level
  - *order of execution*: batch
  - *control strategy*: integrative

- **Crainic et al's taxonomy**:
  - *search control cardinality*: 1-C
  - *search control and communications*: the categories in this dimension are not well fit to this strategy. This would be classified in an intermediate class between KS (the communication is synchronous and its frequency is previously determined) and KC (the suitable number of copies for each operator is inferred from the information sent by the MSA's)
  - *search differentiation*: SPDS

6.4 Experimental framework

The validation of our proposal will be done over instances of the well known SAT problem because of the wide variety of available instance classes and the existence of very simple ad-hoc high performance heuristics that can be used as operators for our strategy. Both issues will take the first two subsections of this part of the chapter.

6.4.1 The SAT problem

The SAT problem is one of the best known decision problems. It has been applied to many fields such as the design and verification of hardware devices, asynchronous circuit design, computer network design and scheduling. An instance of SAT is defined by the following elements:

- A finite set of boolean variables $X = \{x_1, \ldots, x_n\}$ that take values in the domain $B = \{0, 1\}$.
- A set of boolean operators $\Psi = \{\wedge, \vee, \neg\}$, where $\wedge$ is the conjunction, $\vee$ the disjunction and $\neg$ the negation.
- A finite set of literals $L = \{x_1, \neg x_1, \ldots, x_n, \neg x_n\}$. A literal can be a variable or its negation.
- A finite set of clauses $C = \{c_1, \ldots, c_m\}$, where a clause is a set of literals connected by $\wedge$ or $\vee$.
- A boolean formula ϕ is composed by a set of clauses linked by $\wedge$ or $\vee$. If all clauses are connected by the conjunction and all literals in every clause are linked by the disjunction, ϕ is in Conjunctive Normal Form (CNF). We can suppose ϕ is a CNF formula without lost of generality, because any boolean formula can be transformed into CNF.

A truth assignment for X is a mapping $a\colon X \to B$. An assignment a satisfies a clause c_i if $\exists l \in c_i$ such that l maps to 1 under a. A model for ϕ is a truth assignment that satisfies every clause in ϕ.

The SAT problem consists in, given a set of boolean variables $X = \{x_1, \ldots, x_n\}$, a set of clauses $C = \{c_1, \ldots, c_m\}$ and a boolean formula ϕ, finding a model for $\phi = \bigwedge_{i=1}^{m} c_i$. When SAT is treated as a optimisation problem, the objective function is given by the number of non-satisfied clauses. Therefore, the aim is to minimize this function. If the boolean formula is satisfiable, the optimum value is zero.

SAT instance benchmarks

The instance classes used for experimentation were selected from the SATLIB library [80]. It is a very well known repository widely used to test SAT solvers. Concretely, we chose those classes that have at least 100 instances. The sets created for experimentation can be seen in Table 6.1, where we show the name given to the group, the type of their instances, parameters regarding the instance generation, number of variables and clauses, as well as the size of the set. Additionally, we defined a 'MIXED' set composed by 25 elements randomly chosen from UF100 (Uniform Random 3-SAT with 100 variables), RTI, BMS and CBS, so there is a total of 100 instances.

Short name	Class	Parameters	Vars	Cls	Size
UF150	Uniform random 3-SAT [80]		150	645	100
FLAT100	Graph colouring [80]	vertices=100, edge=239	300	1117	100
RTI	Random Three Instances [137]		100	429	500
BMS	Backbone Minimal Sub-instance [137]		100	varies	500
CBS	Controlled Backbone size [137]	backbone size=90	100	403	1000
SW100	SAT-encoded "Morphed" Graph Colouring [61]	vertices=100, edges=400,p= 2^{-4}	500	3100	100
MIXED	25 instances from UF100, CBS, RTI and BMS		100	varies	100

Table 6.1: Characteristics of the benchmark datasets.

6.4.2 SAT specific operators

Stochastic local searches have been widely used to solve SAT problems. They usually start from a randomly generated variable assignment and make local movements by flipping one variable until a model for the formula is found or the stopping condition is fulfilled. The objective function used for these methods is the number of unsatisfied clauses. The majority of these local search algorithms differs in how they select the variable to flip. We have used these variable-flip selection mechanisms to implement the operators in our strategy. All of them are well-known incomplete SAT solvers with a high performance on a broad number of instances of this problem. Concretely, such variable selection mechanisms and the algorithms where they are implemented are:

- **gsat/tabu [109]:** This mechanism derives from gsat [135], another variable selection heuristic that flips the one that produces a bigger decrease in the number of unsatisfied clauses, breaking the ties randomly . Gsat/tabu adds a mechanism to ensure that if a variable has been flipped, then its cannot be flipped back within the next t steps, where t is named "tabu tenure".

algorithm	parameters
wsat	$wp = 0.5$
wsat/tabu	$wp = 0$, $tenure = 3$ FLAT100 instances
	$tenure = 5$ rest of instances
gsat/tabu	$tenure = 10$ FLAT100 instances
	$tenure = 20$ rest of instances

Table 6.2: Parameter settings of the operators

- **wsat [134]:** This heuristic picks randomly one of the unsatisfied clauses c_j. If there is a variable x_i in c_j that does not falsify any clause when is flipped then select x_i. Else, with probability wp flip a random x_i in c_j and with probability $1 - wp$ flip the variable that makes false the smallest number of unsatisfied clauses.

- **wsat/tabu [110]:** As wsat, if a variable x_i from the clause c_j does not falsify any clause when is flipped, such variable is selected. However, when this condition is not fulfilled by any one, only those variables flipped since t or more steps ago are taken into account to be chosen in the else part of the wsat selection strategy.

The operators' parameters are set as shown in Table 6.2. The values are chosen following those presented in [79], where these SAT solvers were applied to three out of the six instance classes used in this chapter (FLAT100, RTI and UF150). For the rest of datasets, we fixed the parameters to those values more frequently used. These coincide with the ones applied by the author to classes RTI and UF150.

In order to implement the tabu mechanism of gsat/tabu and wsat/tabu, we defined a vector of integers for each solution where the time (number of evaluations done from the beginning of the search) of the last modification for each variable is stored.

6.4.3 Further details of the cooperative strategy

In order to test our strategy, a number of implementation decisions have been taken.

Firstly, we decided that for every instance, we will have 6 solutions. The number of agents that integrate $MSAPop$ is also fixed to 6, so the same

MSA is applied to the same solution between two consecutive executions of the third layer.

Regarding the credit assignment definition, let's suppose we have a solution s from instance j, and then a MSA is applied to obtain a new solution $s\prime$. The credit gained will be:

$$\Delta credit(s, s\prime) = \begin{cases} 1, & \text{if } f_j(s\prime) = 0 \\ 0, & \text{otherwise} \end{cases} \tag{6.6}$$

In this way, an agent is only rewarded when the instance is solved. In this chapter, the meaning of "solved" is that all the clauses were satisfied. Function f_j is the objective function for the solutions of the j^{th} instance. The acceptance criterion is $G_{accept}(s, s\prime) = TRUE$, that is, the current solution is always replaced by the new one.

We should remark that this credit assignment definition is well suited for decision problems like SAT. Nevertheless, the whole strategy can be applied to optimisation problems by just considering two aspects: a) a proper definition of the credit assignment function, where the alternatives described in [139] can be explored; and b) when to trigger the operator adjustment and learning mechanism. In this sense, simple rules like "every certain number of iterations of the first layer" could be used.

Three operators are available for our strategy ($|\Theta| = 3$), which corresponds to the variable selection mechanisms used in gsat/tabu, wsat and wsat/tabu, respectively.

The metaheuristic has been coded in Java$^{\text{TM}}$ 1.6. We should also mention that LiO library [108] was used to implement some aspects of the strategy.

6.5 Results

The computational experiments done are oriented to check if given the same amount of resources, our strategy (using embedded SAT solvers), can use them in a more effective and efficient way than such SAT specific algorithms. For this comparison, we will take into account the number of solved instances and how fast they can solve them.

Seven algorithms will be compared:

- Three specific SAT solvers: wsat, wsat/tabu and gsat/tabu.

- Four configurations of our strategy
  1. NAS: sequential with no adaptation (learning off)
  2. AS: sequential with adaptation (learning on)
  3. NAP: simultaneous with no adaptation (learning off)
  4. AP: simultaneous with adaptation (learning on)

We performed 30 runs per algorithm and dataset (shown in Table 6.1) and we recorded the number of non-solved instances as well as the number of evaluations performed [1]. Before each run, the instances are randomly shuffled in order to avoid potential biases in the results due to a particular order. The stopping condition was set to 12×10^7 objective function evaluations for all data sets. This number is very large to allow the solution of all the instances in the datasets (recall that one of them has 1000 instances)

The reader should note that NAS and NAP can be considered as Portfolio approaches in the sense described in [82].

The first analysis relates with the efficacy of every method. We define a single run as successful if all the instances in the dataset were solved. Table 6.3 shows the number of successful runs and the average percentage of non-solved instances by each method in each dataset.

Focusing first in the Global column, we can observe that the number of successful runs is much lower for SAT-specific solvers than for any of our methods. The efficacy of our strategy using the parallel operational mode is excellent: all the instances were solved in all the runs (just 1 run was not successful for AP). Moreover, even a sequential processing of the instances allows to obtain better results. It is also interesting to note that there exists a particular dataset where every SAT-specific solver completely failed to achieve even one successful run.

An interesting point here is to consider how "bad" are those non-successful runs. In other words, what is the average number of non-solved instances. Again, we can observe in Table 6.3, under the Global column, that the values for our strategy are equal or lower than 0.02% which means that in

[1] The reader should note that the computational time per evaluation or local search step is longer for gsat/tabu than for wsat and wsat/tabu. Nevertheless, we decide to use the number of local search steps as an efficiency measure since it has been used in very known works, as [79, 133]. A detailed study about the CPU time per step for these SAT solvers can be found in [79].

algorithm	BMS		CBS		FLAT100		MIXED	
	Runs	Inst	Runs	Inst	Runs	Inst	Runs	Inst
wsat	26	0.03	28	0.01	30	0.00	30	0.00
wsat/tabu	18	0.13	30	0.00	26	0.13	30	0.00
gsat/tabu		21.37	26	0.02	30	0.00	25	0.20
NAS	22	0.07	30	0.00	30	0.00	29	0.03
AS	22	0.05	30	0.00	30	0.00	30	0.00
NAP	30	0.00	30	0.00	30	0.00	30	0.00
AP	30	0.00	30	0.00	30	0.00	30	0.00
	RTI		SW100		UF150		Global	
	Runs	Inst	Runs	Inst	Runs	Inst	Runs	Inst
wsat	30	0.00		36.70	30	0.00	174	5.25
wsat/tabu	29	0.01		26.17	30	0.00	163	3.78
gsat/tabu	30	0.00	30	0.00	28	0.07	169	3.09
NAS	30	0.00	30	0.00	30	0.00	201	0.01
AS	30	0.00	28	0.10	30	0.00	200	0.02
NAP	30	0.00	30	0.00	30	0.00	210	0.0
AP	30	0.00	29	0.03	30	0.00	209	0.01

Table 6.3: Number of successful runs (all instances in the data set are solved) and average percentage of non-solved instances for each method over each dataset.

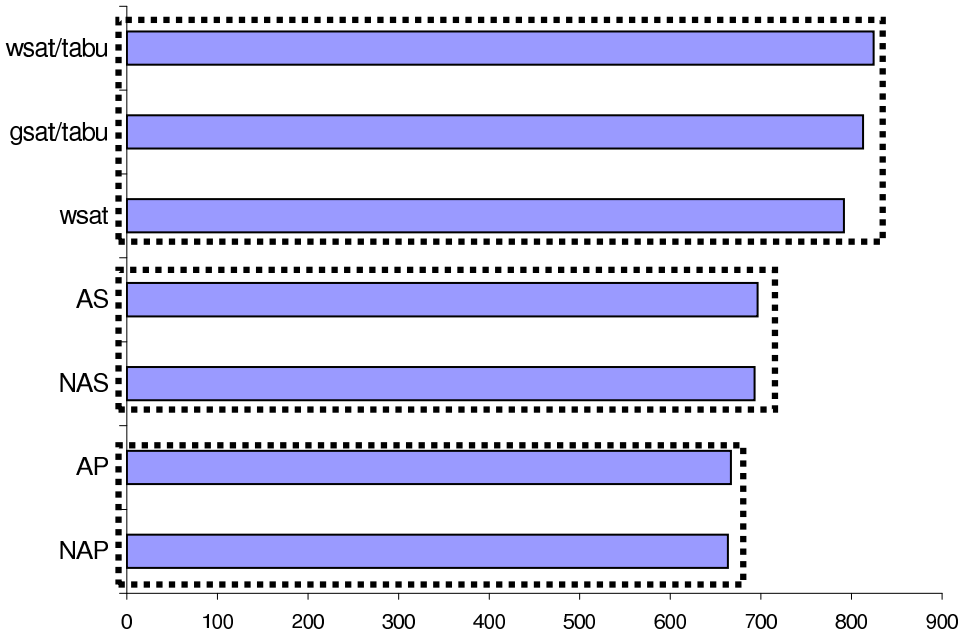


Figure 6.6: Each bar represents the average rank returned by the Kruskal-Wallis non-parametric test for the number of non-solved instances for the different methods considering all datasets. Algorithms in the Y axis are sorted by this value (since we are minimizing, the lower the better). The performance of the algorithms grouped by a dotted rectangle is not significantly different (U Mann-Whitney non-parametrical test with $\alpha < 0.05$)

the non-successful runs, most of the instances could be optimally solved. On the contrary, for SAT specific algorithms, these percentages are much higher, varying from 3.09% for gsat/tabu to 5.25% for wsat.

If we separate these results in terms of the datasets, we should highlight the variance in performance experienced by SAT specific solvers versus the more robust behaviour of our strategy, whose average percentage is zero or a value very near to zero for all cases. The gsat/tabu solver has a good efficacy in 5 out of 6 different datasets, showing bad results in the BMS case, where it could not solve around 21% of the instances. Something similar happens with wsat and wsat/tabu, although for these solvers, the worst performance was in the SW100 dataset, where the average percentages of non-solved instances were 36.70 and 26.17, respectively.

In order to confirm if the observed differences in the means of non-solved instances have statistical significance we have conducted non parametric test

analysis. Figure 6.6 shows a bar plot where the bars represent the average rank returned by the Kruskal-Wallis non-parametric test. The algorithms displayed in the category axis are sorted by this value (since we are minimising, the lower the better). Dotted rectangles group algorithms whose performance is not significantly different (U Mann-Whitney non-parametrical test). Three groups are clearly observed. The most effective algorithms are NAP and AP, followed by NAS and AS, whereas the SAT solvers are the worst taking into account this measure.

Regarding our strategy, it is clear that in terms of efficacy, the role of the operational mode (sequential vs. simultaneous) is more important than learning. Both AP and NAP are significantly better than AS and NAS.

From the point of view of the efficiency, i.e. the number of evaluations required to solve all the instances, the results are presented in Table 6.4 where for each dataset and algorithm, the average, median and standard deviation of the number of evaluations over all runs is shown. Averages over all data sets are not shown as they are distorted by non-successful runs that finish after the limit of evaluations is reached (see for example the differences between the mean and the median in datasets FLAT100 and SW100 for the methods wsat/tabu and AS respectively). The global statistical analysis is displayed in Figure 6.7. In this plot, we can check that there are no significant differences among SAT specific solvers, that is, from the point of view of the average performance over all datasets is indifferent to choose one of these three algorithms. Regarding the multi-instance methods, gsat and wsat are significantly worse than all of them, whereas for wsat/tabu, can achieve a performance equivalent to NAP. This graphic also clearly shows that the improvement obtained by the adaptation mechanism is statistically significant for both AS and AP. Finally, another important aspect is that there are no significant differences between the simultaneous and sequential operational modes.

Figures 6.8, 6.9, 6.10 and 6.11 complements Table 6.4 and shows one plot per dataset with the information of the statistically significant differences. We should first highlight the performance variation for SAT solvers as a

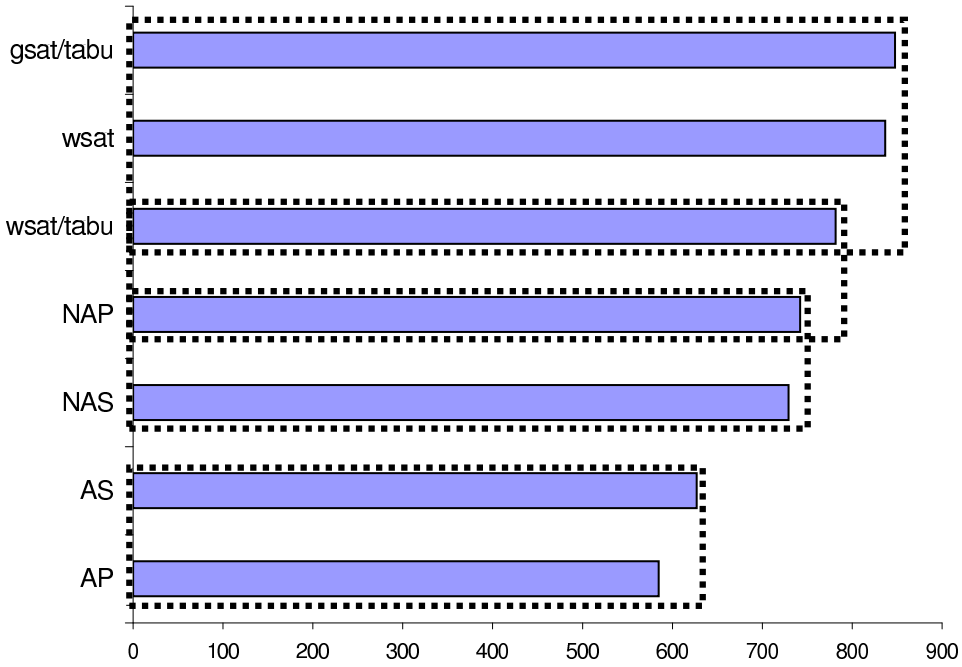


Figure 6.7: Each bar represents the average rank returned by the Kruskal-Wallis non-parametric test for the number of evaluations done by the different methods considering all datasets. Algorithms in Y axis are sorted by this value (since we are minimising, the lower the better). The performance of the algorithms grouped by a dotted rectangle is not significantly different (U Mann-Whitney non-parametrical test with $\alpha < 0.05$)

algorithm	BMS			CBS		
	average	median	std. dev	average	median	std. dev
wsat	$3{,}6\times10^{7}$	$3{,}6\times10^{7}$	$3{,}0\times10^{6}$	$1{,}5\times10^{7}$	$1{,}5\times10^{7}$	$6{,}9\times10^{5}$
wsat/tabu	$4{,}5\times10^{7}$	$4{,}4\times10^{7}$	$5{,}4\times10^{6}$	$1{,}0\times10^{7}$	$1{,}0\times10^{7}$	$5{,}9\times10^{5}$
gsat/tabu	$1{,}1\times10^{8}$	$1{,}1\times10^{8}$	$3{,}8\times10^{5}$	$4{,}5\times10^{7}$	$4{,}5\times10^{7}$	$2{,}4\times10^{6}$
NAS	$4{,}9\times10^{7}$	$4{,}9\times10^{7}$	$3{,}2\times10^{6}$	$1{,}5\times10^{7}$	$1{,}5\times10^{7}$	$7{,}2\times10^{5}$
AS	$3{,}9\times10^{7}$	$3{,}9\times10^{7}$	$4{,}4\times10^{6}$	$1{,}1\times10^{7}$	$1{,}1\times10^{7}$	$6{,}3\times10^{5}$
NAP	$5{,}0\times10^{7}$	$5{,}0\times10^{7}$	$4{,}4\times10^{6}$	$1{,}4\times10^{7}$	$1{,}4\times10^{7}$	$6{,}7\times10^{5}$
AP	$4{,}3\times10^{7}$	$4{,}2\times10^{7}$	$4{,}9\times10^{6}$	$1{,}1\times10^{7}$	$1{,}1\times10^{7}$	$6{,}4\times10^{5}$
algorithm	FLAT100			RTI		
	average	median	std. dev	average	median	std. dev
wsat	$4{,}2\times10^{6}$	$4{,}2\times10^{6}$	$5{,}5\times10^{5}$	$1{,}7\times10^{6}$	$1{,}7\times10^{6}$	$1{,}8\times10^{5}$
wsat/tabu	$2{,}1\times10^{7}$	$4{,}3\times10^{6}$	$4{,}4\times10^{7}$	$1{,}1\times10^{6}$	$1{,}1\times10^{6}$	$1{,}4\times10^{5}$
gsat/tabu	$1{,}5\times10^{6}$	$1{,}5\times10^{6}$	$2{,}5\times10^{5}$	$4{,}9\times10^{6}$	$4{,}8\times10^{6}$	$4{,}7\times10^{5}$
NAS	$2{,}8\times10^{6}$	$2{,}7\times10^{6}$	$4{,}3\times10^{5}$	$1{,}8\times10^{6}$	$1{,}8\times10^{6}$	$1{,}9\times10^{5}$
AS	$1{,}9\times10^{6}$	$1{,}9\times10^{6}$	$3{,}0\times10^{5}$	$1{,}4\times10^{6}$	$1{,}4\times10^{6}$	$1{,}9\times10^{5}$
NAP	$2{,}8\times10^{6}$	$2{,}8\times10^{6}$	$3{,}5\times10^{5}$	$1{,}8\times10^{6}$	$1{,}8\times10^{6}$	$1{,}2\times10^{5}$
AP	$2{,}1\times10^{6}$	$2{,}1\times10^{6}$	$3{,}5\times10^{5}$	$1{,}3\times10^{6}$	$1{,}3\times10^{6}$	$1{,}3\times10^{5}$
algorithm	SW100			UF150		
	average	median	std. dev	average	median	std. dev
wsat	$1{,}1\times10^{8}$	$1{,}1\times10^{8}$	$2{,}2\times10^{6}$	$1{,}3\times10^{6}$	$1{,}3\times10^{6}$	$2{,}9\times10^{5}$
wsat/tabu	$1{,}1\times10^{8}$	$1{,}1\times10^{8}$	$4{,}5\times10^{6}$	$1{,}3\times10^{6}$	$1{,}2\times10^{6}$	$4{,}5\times10^{5}$
gsat/tabu	$1{,}2\times10^{6}$	$1{,}2\times10^{6}$	$1{,}1\times10^{5}$	$2{,}4\times10^{6}$	$2{,}1\times10^{6}$	$8{,}1\times10^{5}$
NAS	$2{,}2\times10^{6}$	$2{,}1\times10^{6}$	$1{,}8\times10^{5}$	$1{,}2\times10^{6}$	$1{,}1\times10^{6}$	$3{,}3\times10^{5}$
AS	$8{,}8\times10^{6}$	$1{,}7\times10^{6}$	$2{,}7\times10^{7}$	$1{,}2\times10^{6}$	$1{,}3\times10^{6}$	$2{,}9\times10^{5}$
NAP	$2{,}2\times10^{6}$	$2{,}2\times10^{6}$	$2{,}8\times10^{5}$	$1{,}4\times10^{6}$	$1{,}2\times10^{6}$	$4{,}3\times10^{5}$
AP	$1{,}6\times10^{6}$	$1{,}5\times10^{6}$	$2{,}0\times10^{5}$	$1{,}2\times10^{6}$	$1{,}2\times10^{6}$	$2{,}4\times10^{5}$

algorithm	MIXED		
	average	median	std. dev
wsat	$2{,}2\times10^{6}$	$2{,}1\times10^{6}$	$8{,}8\times10^{5}$
wsat/tabu	$2{,}3\times10^{6}$	$1{,}9\times10^{6}$	$1{,}4\times10^{6}$
gsat/tabu	$2{,}8\times10^{7}$	$2{,}2\times10^{7}$	$1{,}7\times10^{7}$
NAS	$2{,}4\times10^{6}$	$2{,}2\times10^{6}$	$1{,}0\times10^{6}$
AS	$2{,}8\times10^{6}$	$2{,}2\times10^{6}$	$1{,}6\times10^{6}$
NAP	$2{,}6\times10^{6}$	$2{,}3\times10^{6}$	$1{,}2\times10^{6}$
AP	$1{,}9\times10^{6}$	$1{,}7\times10^{6}$	$1{,}0\times10^{6}$

Table 6.4: Average, median and standard deviation of the number of evaluations done by the different algorithms over each dataset.

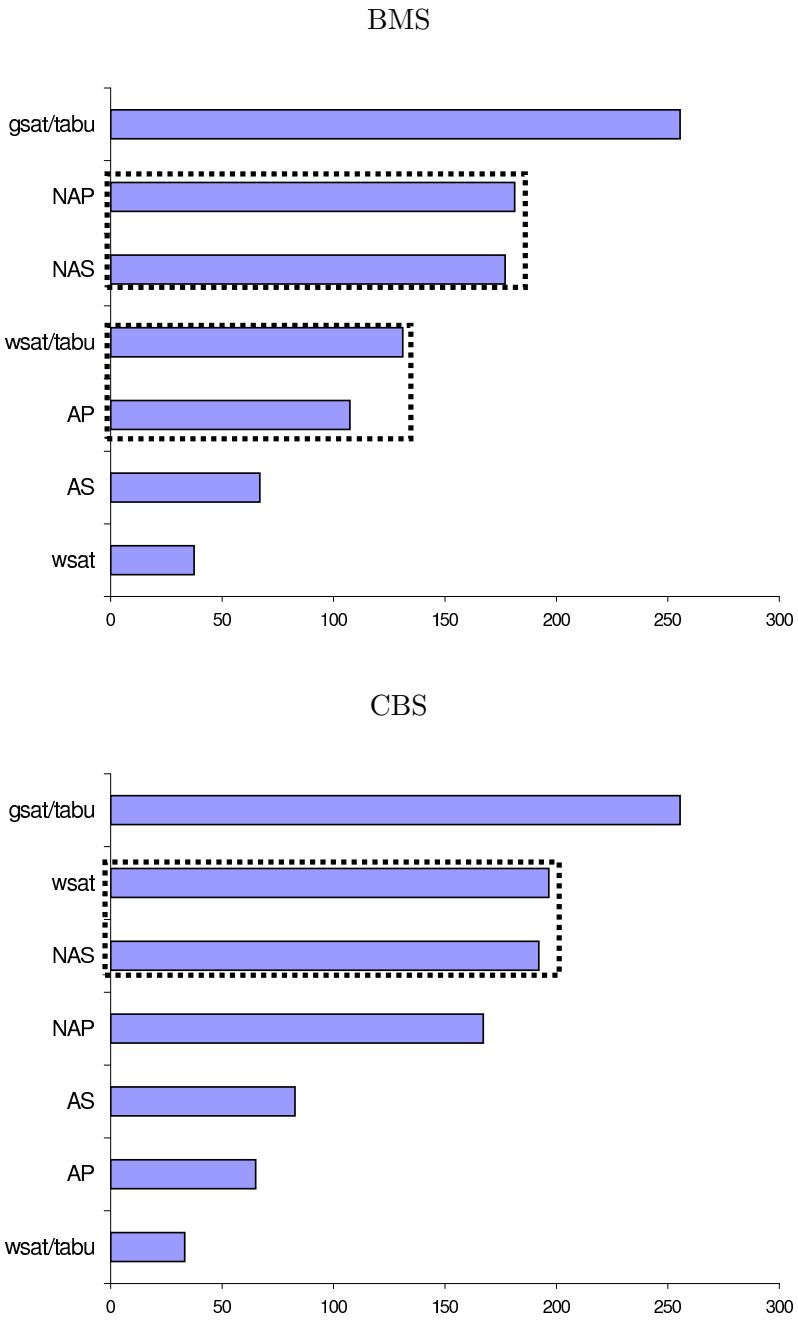


Figure 6.8: Each bar represents the average rank returned by the Kruskal-Wallis non-parametric test for the number of evaluations done by the different methods over each dataset. Algorithms in Y axis are sorted by this value (since we are minimising, the lower the better). The performance of the algorithms grouped by a dotted rectangle is not significantly different (U Mann-Whitney non-parametrical test with $\alpha < 0.05$)

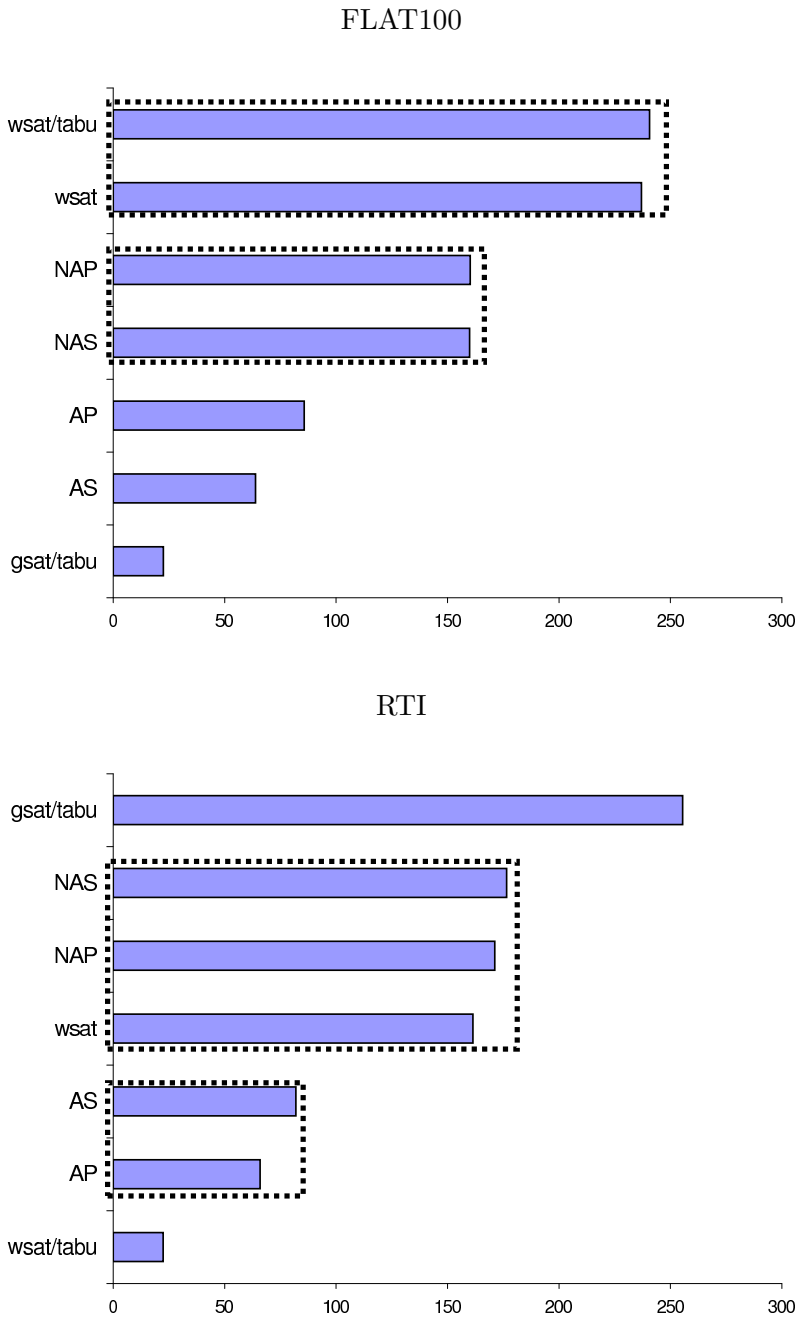


Figure 6.9: Each bar represents the average rank returned by the Kruskal-Wallis non-parametric test for the number of evaluations done by the different methods over each dataset. Algorithms in Y axis are sorted by this value (since we are minimising, the lower the better). The performance of the algorithms grouped by a dotted rectangle is not significantly different (U Mann-Whitney non-parametrical test with $\alpha < 0.05$)

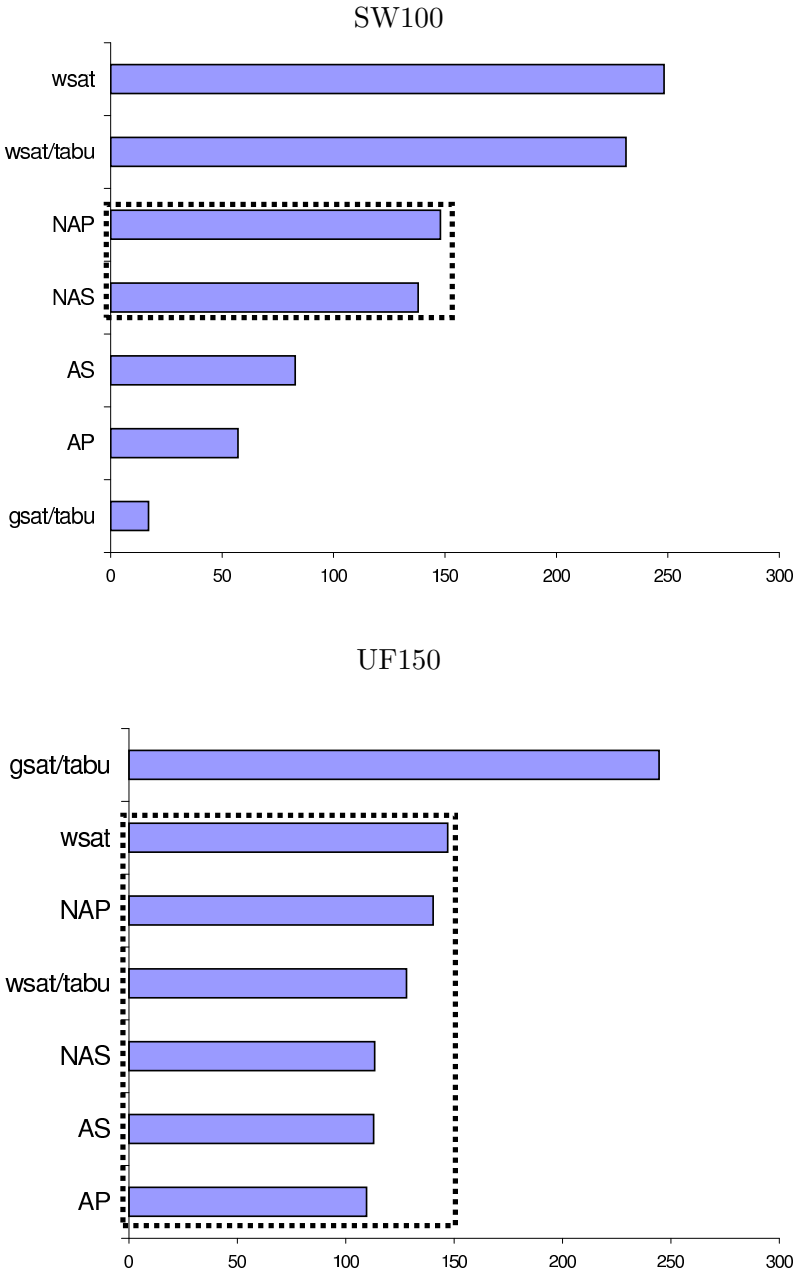


Figure 6.10: Each bar represents the average rank returned by the Kruskal-Wallis non-parametric test for the number of evaluations done by the different methods over each dataset. Algorithms in Y axis are sorted by this value (since we are minimising, the lower the better). The performance of the algorithms grouped by a dotted rectangle is not significantly different (U Mann-Whitney non-parametrical test with $\alpha < 0.05$)

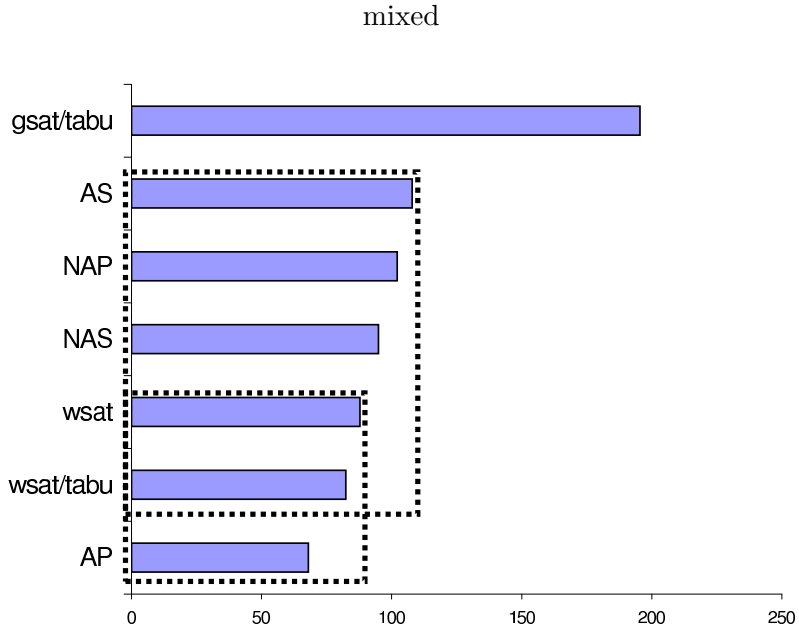


Figure 6.11: Each bar represents the average rank returned by the Kruskal-Wallis non-parametric test for the number of evaluations done by the different methods over each dataset. Algorithms in Y axis are sorted by this value (since we are minimising, the lower the better). The performance of the algorithms grouped by a dotted rectangle is not significantly different (U Mann-Whitney non-parametrical test with $\alpha < 0.05$)

function of the dataset. Wsat is significantly the faster method for the BMS set, but the worst in FLAT100 and SW100. Wsat/tabu is the most efficient algorithm for CBS and RTI instances, whereas its performance drops until the last position in FLAT100. Gsat/tabu shows even more extreme behavior changes: it is significantly the fastest algorithm in FLAT100 and SW100 but the slowest in the rest of datasets. This low level of robustness contrasts with the one experienced by AS and AP, which are more robust along the different datasets. Although none of these methods are ranked in first place (with the exception of AP in UF150 and MIXED datasets), it can be observed that they are always among the three best alternatives.

Focusing now in the features of our strategy, operational mode and learning on/off, we can clearly assert that both operational modes (sequential and parallel) required the same amount of resources if learning is not used: NAS and NAP always appear within a dotted rectangle (with the exception of CBS dataset). When learning is activated, the strategy becomes more efficient: AS is better than NAS and AP is better than NAP. The differences between AS and AP are not so clear. The fist one is significantly better in BMS and FLAT100 datasets, while the later is better in CBS, SW100 and MIXED. However, we should recall that AP could solve all the instances in all the runs, thus meaning that both, the parallel processing and learning lead to a strategy that is both effective and efficient.

It is interesting to remark here the results for the MIXED dataset, which may be considered the worst scenario for the simple adaptation mechanism proposed. In some sense, the results here can be interpreted as the average behavior of the methods over SAT because it contains a showcase of the several types of instances that may appear. The strategy using learning and the simultaneous processing of the instances is ranked first, thus confirming the benefits of the two essential components of our proposal.

Although this is not our aim, we would like to point out that competitiveness with state-of-the-art methods like [155] can not be claimed.

6.5.1 Analysing the dynamic behaviour of the strategy

The previous results confirmed the benefits of our proposal but they do not shed any light on how they are achieved. In this subsection we will study the dynamic behaviour of our strategy from two different aspects. On

one hand, we will see how the number of solved instances varies during the execution for specific SAT solvers, AS and AP. On the other hand, we will also study the dynamic working of the adaptation mechanism using the so called concentration plots, to visualise the evolution of the number of copies for each operator when both operational modes are considered.

The number of solved instances as a function of the time can be associated with the "throughput" of the strategy. If we recall the idea of a server processing several clients' requests, then we will analyse a measurement equivalent to the number of clients served as a function of time.

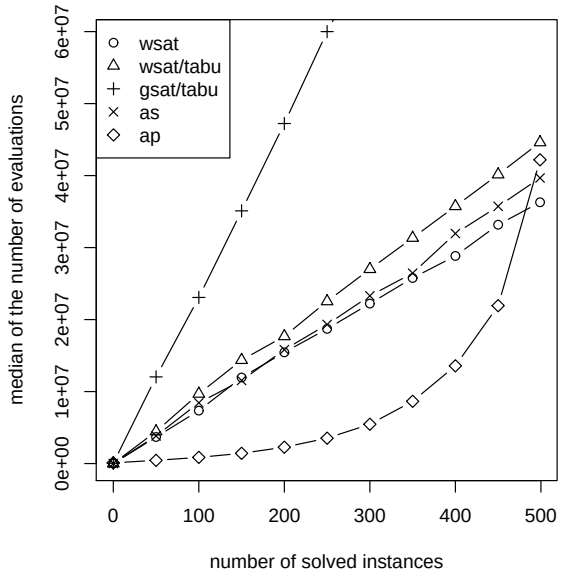
Number of solved instances as a function of time

Figures 6.12, 6.13, 6.14 and 6.15 shows for each algorithm and dataset, the median value of the number of evaluations needed to solve certain number of instances. Recall that the instances are shuffled in every run so the value is not affected by a particular instances' order.

It is easy to observe the variable behaviour of SAT-solvers when different datasets are considered. In general, the worst alternative seems to be gsat/tabu, although it becomes the best alternative for SW100 and FLAT100 datasets. Wsat is worse than wsat/tabu in CBS dataset, while it is better in BMS. The three SAT-Solvers and the AS scheme show a linear tendency but with different slope. However, in terms of throughput, the AS strategy behaviour is quite similar to the one observed for the best SAT-solver in every dataset (although this best one is not always the same).

Setting apart datasets SW100 and FLAT100 where the benefits are not so clear, the AP strategy (using parallel mode and learning activated) is able to solve more instances in less time. In terms of throughput, AP needs a lower time to serve up to 90 percent of the requests. The reason is simple. The parallel resolution, implicitly, solves the easier instances in first place, while the most complex ones are left for the final stages of the search process; this fact explains the tendency change experienced by the AP curve. However, when the solving process is carried out sequentially, the easy and hard instances are mixed, and hence, their results can not be known until their corresponding due times are reached. Another interesting aspect we

BMS



CBS

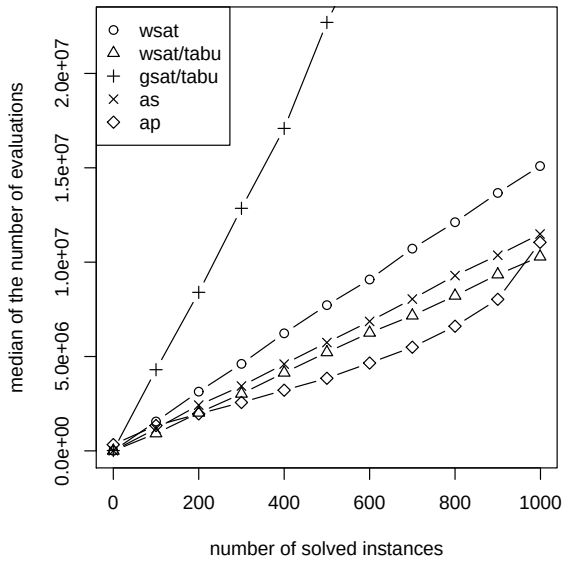


Figure 6.12: Evolution of the median value of the number of evaluations required to solve a determined number of instances. Some series were omitted for visualisation purposes.

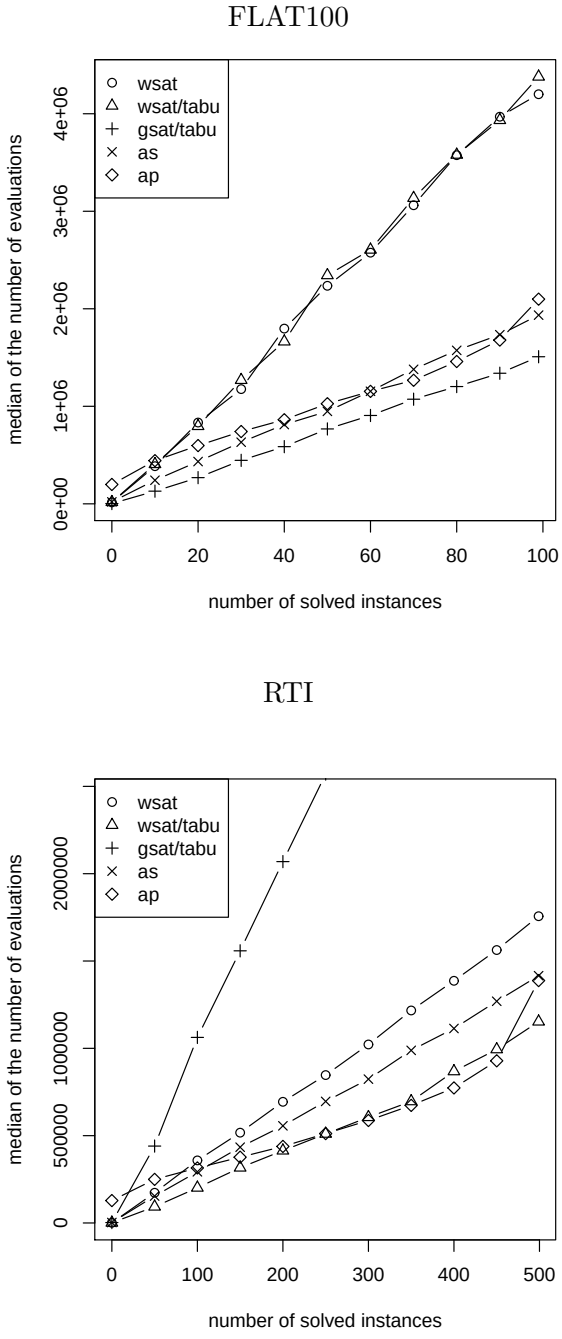
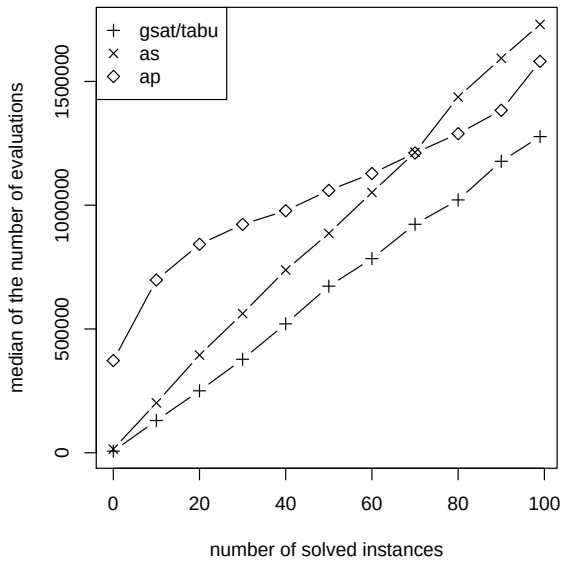


Figure 6.13: Evolution of the median value of the number of evaluations required to solve a determined number of instances. Some series were omitted for visualisation purposes.

SW100



UF150

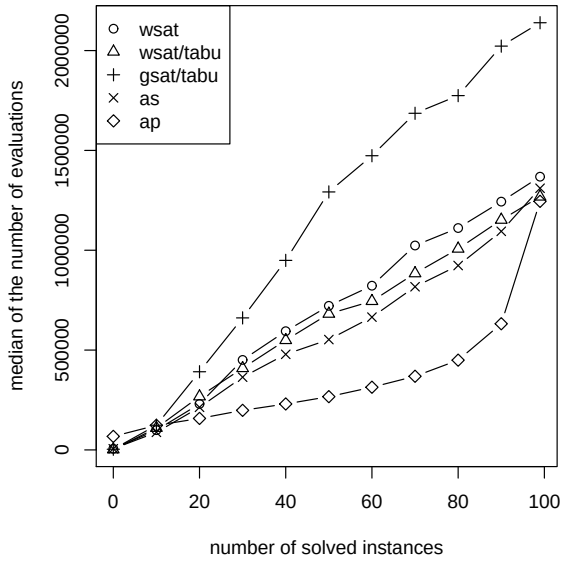


Figure 6.14: Evolution of the median value of the number of evaluations required to solve a determined number of instances. Some series were omitted for visualisation purposes.

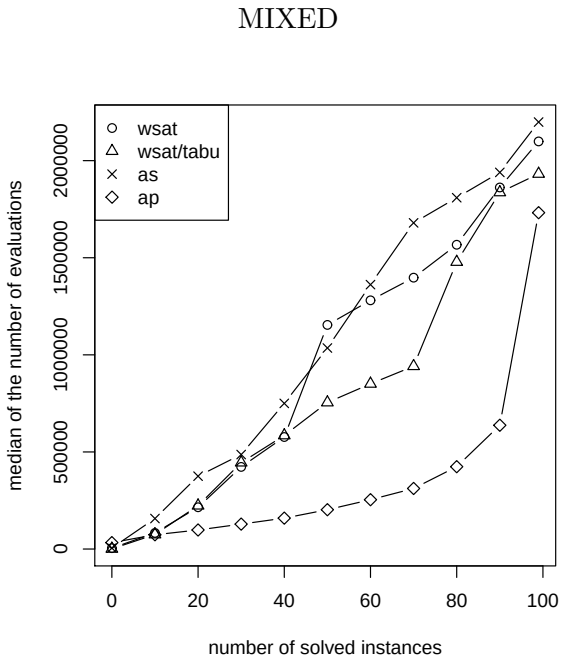


Figure 6.15: Evolution of the median value of the number of evaluations required to solve a determined number of instances. Some series were omitted for visualisation purposes.

can observe is that the benefit is not obvious if a low number of instances is considered. AP needs a higher time to start solving instances, due to the round robin processing of all the elements in the dataset, whereas AS and the SAT solvers concentrate such effort in just one instance.

Once again we should highlight the results in the MIXED dataset. In this hardest case, the AP strategy has enough potential to manage different types of instances efficiently and much better than the other alternatives. If we consider the time needed to solve 90 instances, AP required around 5×10^5 evaluations while the other methods almost needed 2×10^6.

On the adaptation of operators

The adaptation mechanism proposed needs to detect which operator or operators are better suited for each dataset. The goal is to assign the most suitable operators in each case to every MSA. The results in the previous section confirm the usefulness of the adaptation mechanism proposed, but here, we want to study how the number of copies of each operator evolves during the run. A way to visualise this, is to use concentration plots as described in [124]. In these plots, we will have a data series for each possible operator $O_k \in \Theta$. Every point in the series represents the average number of MSAs that are working with operator O_k at equally spaced intervals.These averages are calculated over at least 15 runs.

Figures 6.16, 6.17, 6.18 and 6.19 shows the concentration plots of some datasets, displaying the average concentration of the operators wsat, wsat/tabu and gsat/tabu for both sequential (empty markers) and simultaneous (filled markers) operational modes.

In some datasets, the evolution of the operator's concentration is the expected one, that is to say, the number of copies for each operator is related to the average performance of its respective SAT solver in that instance type. Such types are CBS, RTI and MIXED where the SAT solver ranking for all of them is wsat/tabu - wsat - gsat/tabu, although the number of copies distribution varies with the operational mode. Concretely, the parallel mode (AP) gives more relevance to wsat/tabu operator, making clear that it can detect faster the good operators.

Some interesting peculiarities can be observed in the other datasets. Focusing first on BMS, both AP and AS assign a bigger number of MSA to wsat/tabu than wsat which contrasts with the results seen in Figure 6.8 where wsat was the best SAT solver for this instance type.

To understand this behaviour we need to study how hard are the instances within a given dataset. We made 30 runs of each SAT specific solver for every instance in those datasets which showed some peculiarities in its behavior, and we recorded the median of the number of evaluations needed to solve it. In this way, we can observe how many instances can be solved using a given number of evaluations. The information is shown on Figures 6.20, 6.21, 6.22 and 6.23. We can observe that although wsat/tabu requires less evaluations to solve more instances (e.g. wsat/tabu solves 250 instances using less than 10000 evaluations, while wsat just solved around 120), there exits a subset of them where this solver shows an extremely poor performance. These hard instances made that, on average, the results were worst than those for wsat in this dataset.

The fact that wsat/tabu solves easier instances faster than wsat also explains why AP behaves worse than AS for BMS. Since the parallel mode tends to solve the easier instances first, the learning/adaptation mechanism promotes a bigger concentration of wsat/tabu operator than the sequential mode. When the harder instances arrive, AP can not "learn" until an instance is solved (when the adaptation mechanism is triggered) and as a consequence, it spends much time using not well suited operators until the good ones are promoted. In the AS case, as the instances are mixed, the learning/adaptation mechanism maintains a different dynamic leading to a diverse set of operators (none of them reached 5 copies).

This "unexpected" distribution of the number of copies is also observed in FLAT100, UF150 and SW100 due to similar reasons. For FLAT100 and UF150, this happens for the two worst solvers. As we can see in Figures 6.22 and 6.23, in both cases the third ranked algorithm is faster for easier instances (thus quickly obtains more copies), but slower than the second one for harder instances. Apart from this, in SW100 we can observe that the parallel mode shows a very interesting behaviour (Fig. 6.18). Concretely, during the first stages of the search process, gsat/tabu is the most rewarded

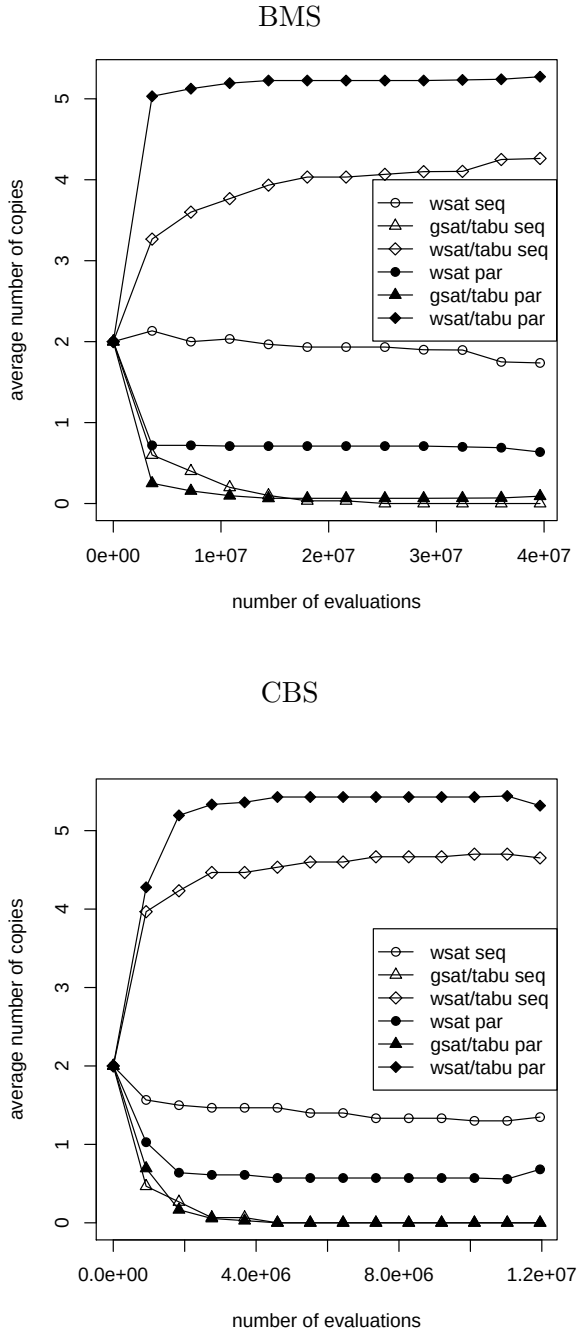
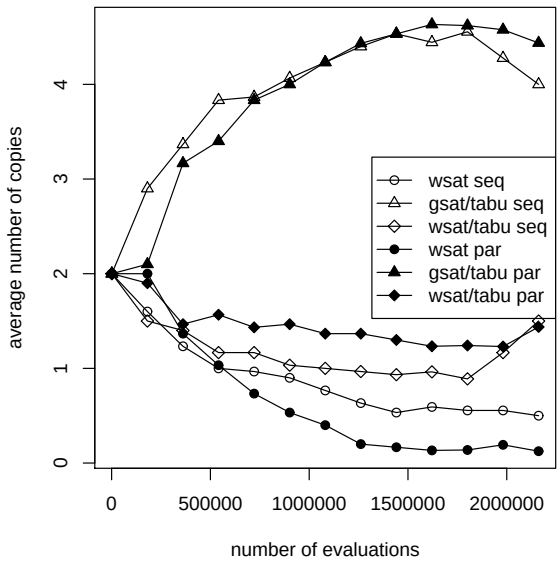


Figure 6.16: Evolution of the number of copies of each operator (wsat, wsat/tabu and gsat/tabu) when sequential (empty markers) and simultaneous (filled markers) operational modes are considered.

FLAT100



RTI

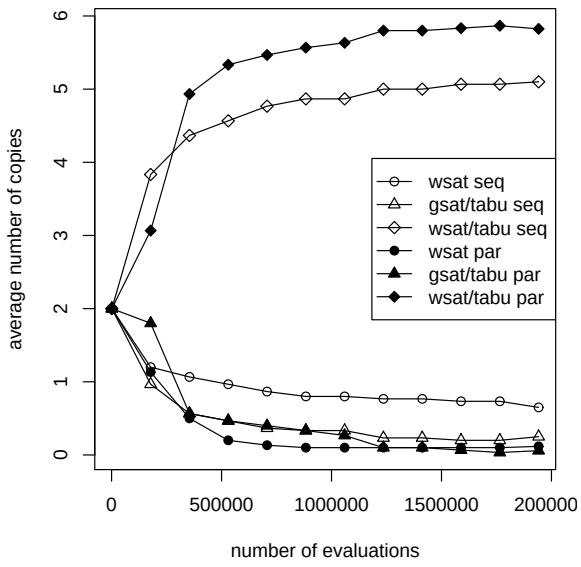


Figure 6.17: Evolution of the number of copies of each operator (wsat, wsat/tabu and gsat/tabu) when sequential (empty markers) and simultaneous (filled markers) operational modes are considered.

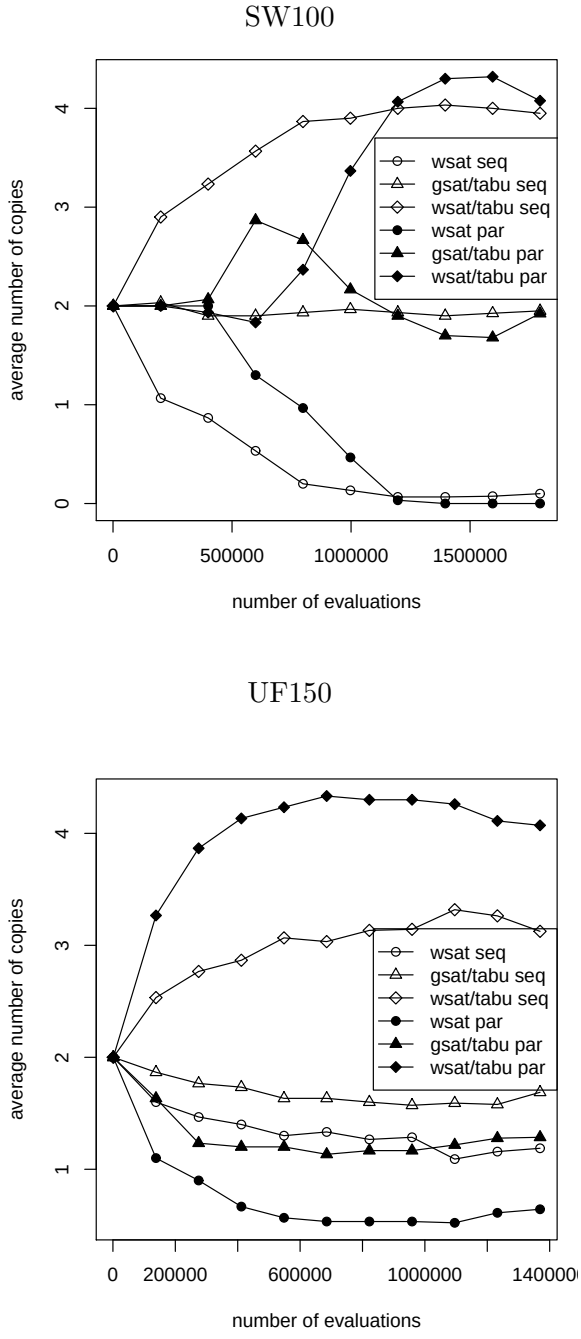


Figure 6.18: Evolution of the number of copies of each operator (wsat, wsat/tabu and gsat/tabu) when sequential (empty markers) and simultaneous (filled markers) operational modes are considered.

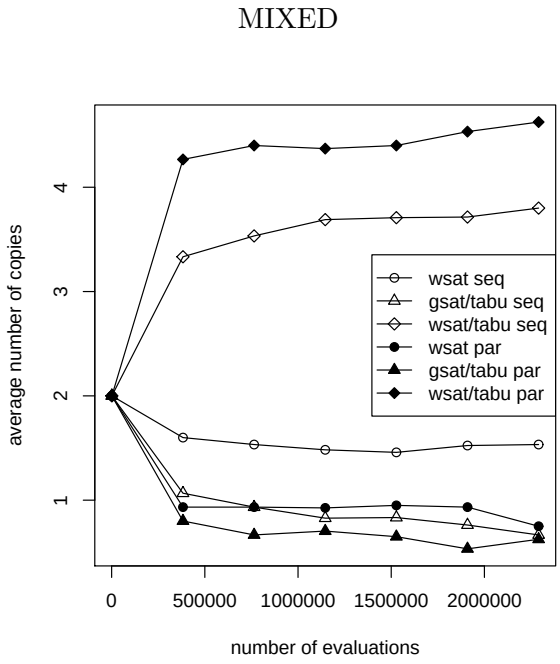


Figure 6.19: Evolution of the number of copies of each operator (wsat, wsat/tabu and gsat/tabu) when sequential (empty markers) and simultaneous (filled markers) operational modes are considered.

BMS

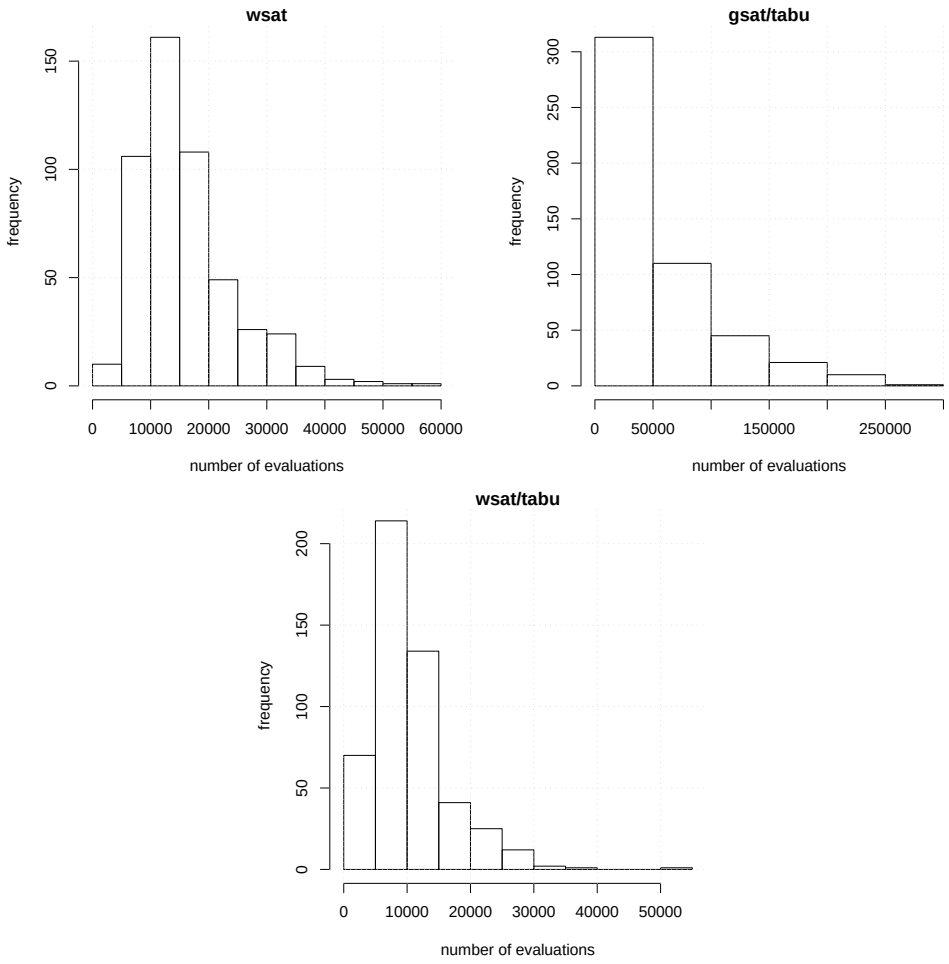


Figure 6.20: Histograms of the median value of the number of evaluations required to solve individual instances in BMS dataset for SAT-specific solvers.

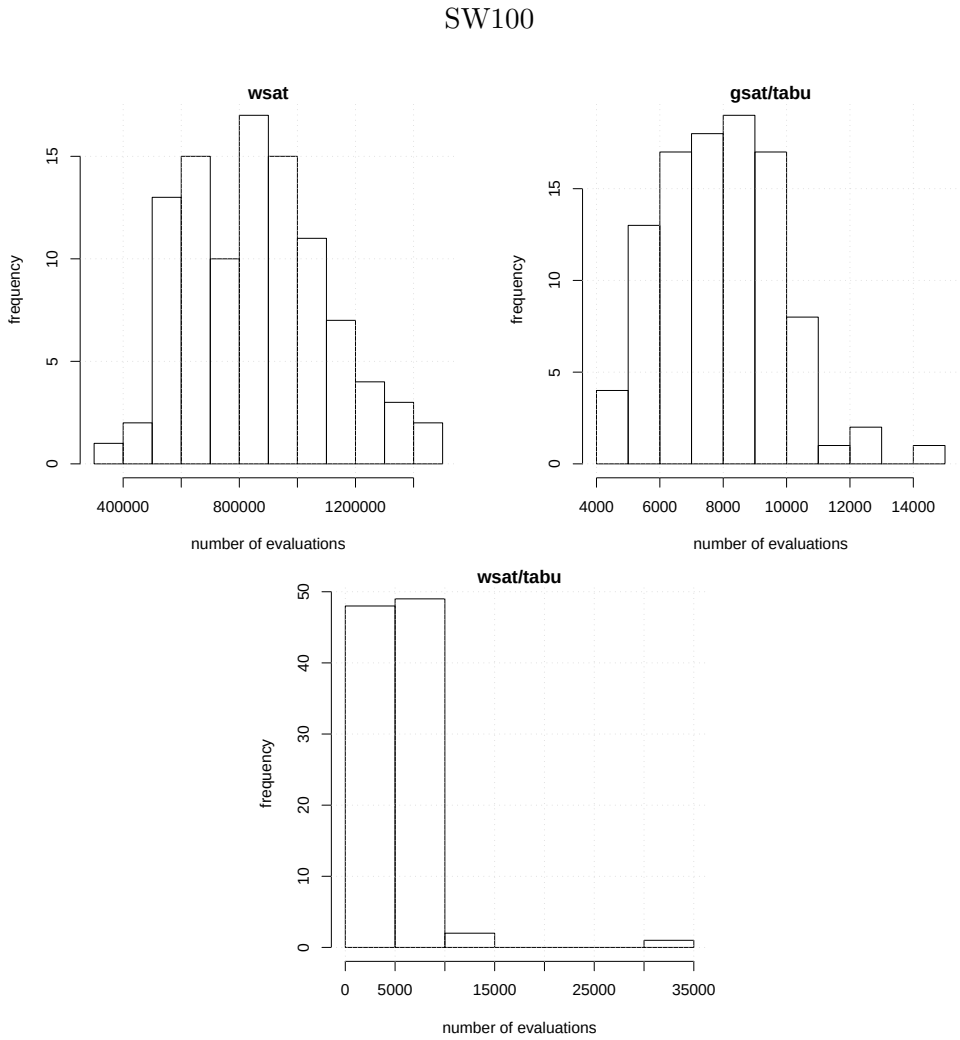


Figure 6.21: Histograms of the median value of the number of evaluations required to solve individual instances in SW100 dataset for SAT-specific solvers.

FLAT100

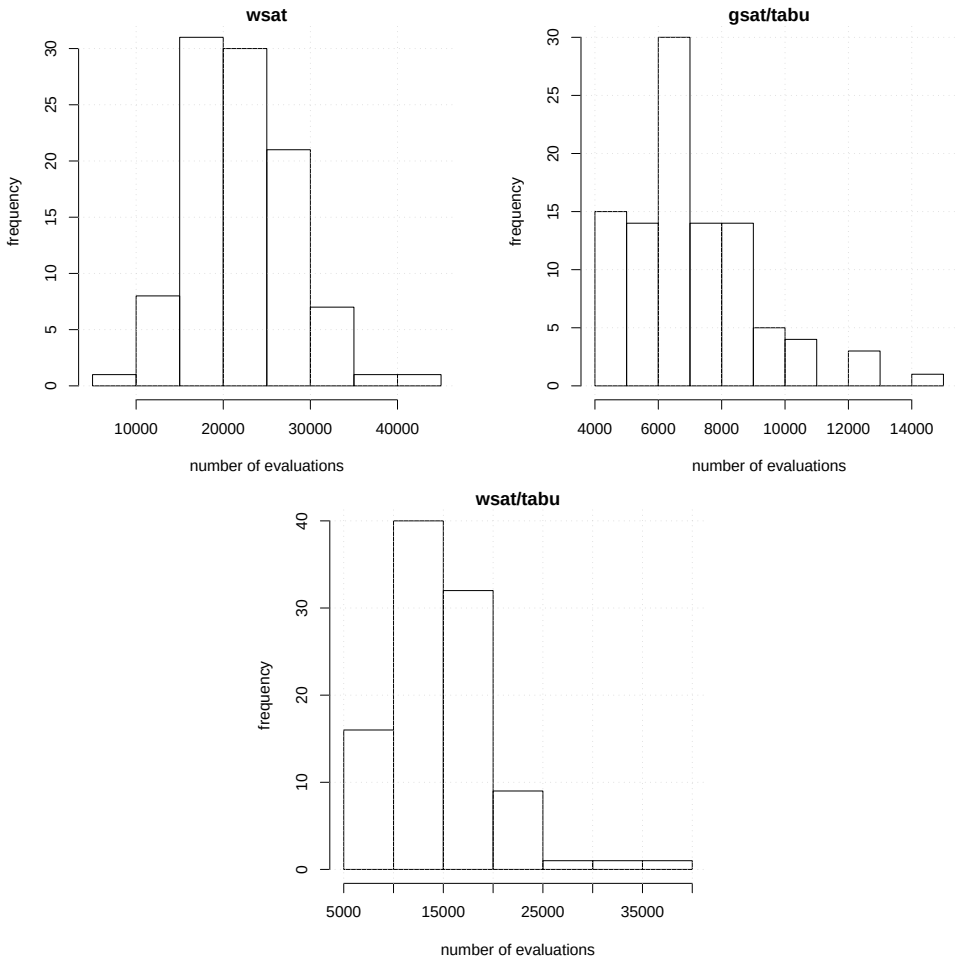


Figure 6.22: Histograms of the median value of the number of evaluations required to solve individual instances in FLAT100 dataset for SAT-specific solvers.

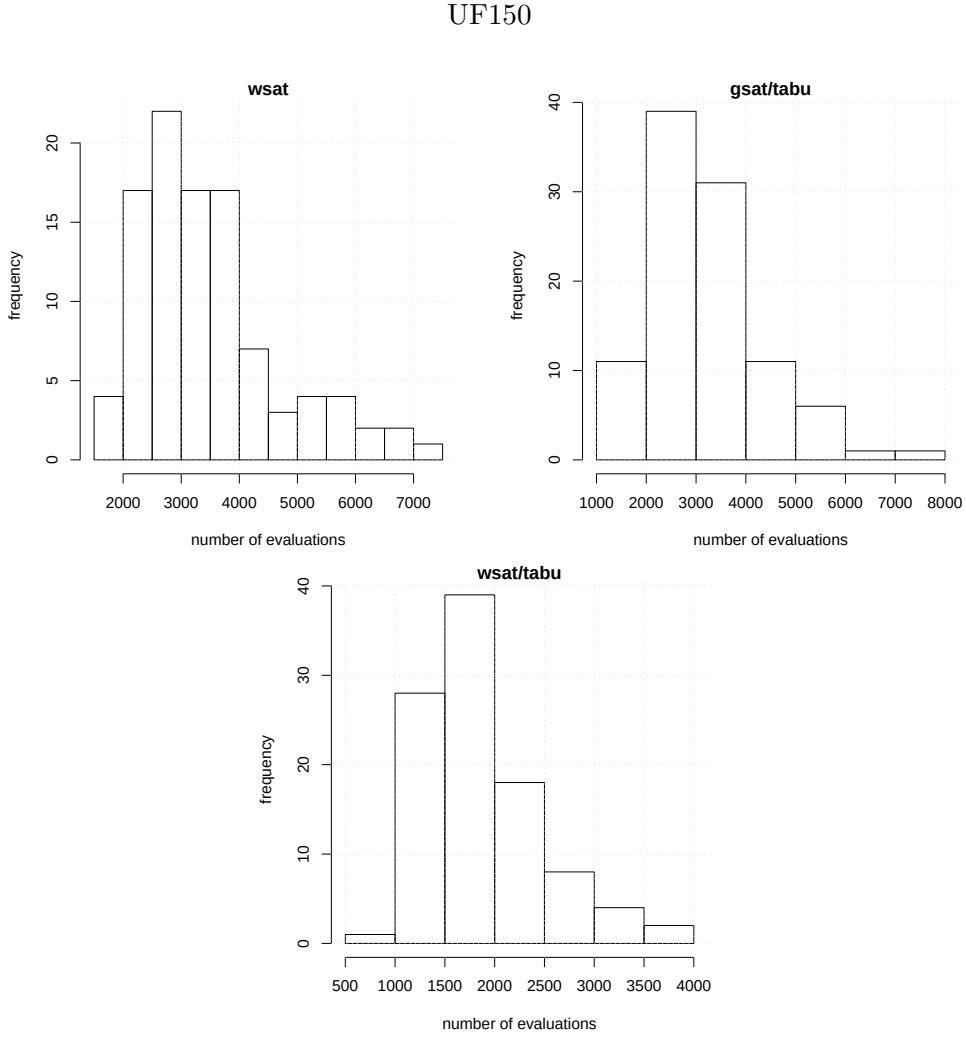


Figure 6.23: Histograms of the median value of the number of evaluations required to solve individual instances in UF150 dataset for SAT-specific solvers.

operator but, after some time, that position is occupied by wsat/tabu. This is more related with the characteristics of the adaptation mechanism than to the heterogeneity among the instances. As the adaptation mechanism deals with an integer number of copies, the copy lost by wsat (due to its bad performance) should be given to one of the other two operators. Since in that initial period gsat/tabu solves slightly more instances than wsat/tabu, it receives that copy. Then, more instances become solved by wsat/tabu and thus, this operator receives more copies.

These results indicate that the adaptation mechanism proposed effectively rewards the most efficient operators at every stage of the run. In general, the mechanism detects the most efficient alternative and quickly assigns most of the copies to it. The rest of the copies are the ones that vary along the run and as the results show, the variation is properly performed.

The role of the adaptation in SAT has been also studied in others works as [13] and [84], but the learning was done in a off-line way. However, in our case, the learning is done on-line due to the fact that we do not know which is the type of instance we need to solve. Apart from this, we can say that both off-line and on-line methods are complementary since these last ones have some fixed parameters that can be tuned by the off-line methods.

6.6 Conclusions

A straight approach to solve a set of instances of a given problem is to sequentially apply a given algorithm over each instance.

In this chapter we have proposed a different strategy to solve a set of instances having in mind the idea that the effort and knowledge gained while solving an instance should be used to solve others. Our strategy is a co-evolving system where solutions for different instances are improved and operators are dynamically scheduled as a function of their performance over the instances that have been already solved. Also, the strategy allows to work in two different operational modes: sequential, where the instances are solved one by one, and simultaneous or parallel, in which all the instances are processed at the same time.

The main conclusion is that using embedded problem specific algorithms, our strategy (specially the AP version) outperforms ad-hoc SAT non-adaptive algorithms both in terms of efficacy (number of solved instances) and effi-

ciency (evaluations of the objective function).

The comparison of the operational modes revealed that the simultaneous case showed a more robust behaviour than the sequential one in terms of efficacy (AP better than AS, and NAP better than NAS) due to the better use of the total amount of resources (evaluations) available. Apart from this, the simultaneous mode shows an additional advantage: it outperforms the sequential mode in terms of solving more instances in less time because the simultaneous operational mode solves first the easiest instances, leaving the hard ones to the last stages of the search. The use of the adaptation and the rewarding mechanism proposed was clearly beneficial. In particular AS was better than NAS, and AP was better than NAP, since they needed to use a lower number of evaluations to solve the same set of instances.

Part IV

Conclusions

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This dissertation has focused on the study, design, development and application of centralised cooperative strategies for optimisation problems. The objectives proposed for this thesis were the following:

1. Make an in-depth study of centralised cooperative strategies.
2. Research on new application areas for this type of methods.
3. Validate the performance of these metaheuristics with respect to state-of-the-art algorithms in a proper manner.
4. Develop and analyse cooperative strategies that can manage the resolution of a set of instances in a effective and efficient way.

With regards to objective 1, departing from a previously developed cooperative strategy where the agents are controlled by a central coordinator that takes decisions basing on a fuzzy rule. We have extended this research in two key points. In first place we studied, in the context of the combinatorial optimisation problems, how the composition of a cooperative strategy can affect its performance. Using as test bed the Uncapacitated Single Allocation p-Hub Median Problem (USApHMP) we analysed the behaviour of the method when all solvers implement the same heuristic (homogeneous strategy) and when each solver implements a different one (heterogeneous

strategy). To have a reference of the performance of such strategies, these were compared with the isolated individual version of the distinct heuristics used as solvers. Concerning this last point, we saw that by means of homogeneous cooperation based on our scheme, the average error of three different metaheuristics can be markedly reduced. We also compared the distinct cooperative strategies studied versus the best individual method which can vary from an instance to another. The obtained results showed that cooperation, both homogeneous one and heterogeneous one, leads to equal to or better average fitness values than the best individual metaheuristic in virtually all cases. The last aspect analysed here was the existing performance differences between the cooperative methods, where it should be highlighted that a) those homogeneous strategies whose solvers implement the best global individual metaheuristic represent the best alternative in terms of performance and b) when the comparison is done between the heterogeneous and the homogeneous composition, we checked that the first one presents some advantages over the second ones. These results were published as a book chapter of the series Studies in Computational Intelligence in [106].

The second point in which we extended the mentioned research was the cooperation scheme. Concretely, we incorporated a control rule based on the reactive search framework [8] into the centralised cooperative strategy. The experiments were also done on instances of the USApHMP and showed that the proposed reactive cooperation scheme achieves better results with respect to a strategy based on independent solvers (where solvers do not exchange information), and with respect to the same cooperative strategy based on the fuzzy control rule used in the former case. We also tested the cooperative strategy using both rules, reactive and fuzzy, at the same time. In this case, the bigger complexity did not pay off because the reactive rule alone performs at least as well as both rules together. Moreover, the reactive rule is able to adapt its behaviour according to the characteristics of the instance, and is effective for detecting stagnation and for driving diversification strategies. These results were included in a book chapter of the series Lecture Notes in Computer Sciences [105] and in an extended abstract on the conference Learning and Intelligent Optimization 2009 (LION 3) [104].

Another aspect tackled in this dissertation was the study of new scenarios where centralised cooperative methods (objective 2) can be applied.

The chosen scenario was Dynamic Optimisation Problems (DOPs) and concretely, those ones whose objective function change over the time. The cooperative strategy developed to deal with these problems had the same structure than the one used formerly. The solvers implemented a trajectory-based algorithm with different parameter configurations. We saw that this type of methods can be easily adapted to DOPs by adding a mechanism that controls the objective function changes and programming the restart of certain memories every time that one of these changes takes place. As far as our knowledge is concerned this kind of trajectory solvers and this scheme of cooperation had not been tested before in DOPs. The method showed very promising results that were able to significantly outperform two state of the art algorithms (Agents and multi-QPSO) in most of the test problems (objective 3). Moreover, it has been believed and assumed that population based methods were best suited to deal with DOPs because it was supposed that a bigger number of solutions could accomplish a better track of a changing environment. While this still may hold true, the success of this centralised cooperative strategy based on trajectory methods could have an important role to play in the dynamic optimization field. An article containing these results is accepted for publication in the Memetic Computing journal [60].

To fulfill the objective 4, we proposed a method based on cooperative strategies to solve a set of instances having in mind the idea that the effort and knowledge gained while solving an instance should be used to solve others. Our strategy is a co-evolving system where solutions for different instances are improved and operators are dynamically scheduled as a function of their performance over the instances that have been already solved. Also, the strategy allows to work in two different operational modes: sequential, where the instances are solved one by one, and simultaneous or parallel, in which all the instances are processed at the same time. The experimentation was done using the SAT problem as benchmark and, apart from three SAT specific solvers, the next four configurations of our strategy were consider:

- NAS: sequential with no adaptation (learning off)
- AS: sequential with adaptation (learning on)
- NAP: simultaneous with no adaptation (learning off)
- AP: simultaneous with adaptation (learning on)

The main conclusion was that using embedded problem specific algorithms, our strategy (specially the AP version) outperforms ad-hoc SAT non-adaptive algorithms both in terms of efficacy (number of solved instances) and efficiency (evaluations of the objective function). The comparison of the operational modes revealed that the simultaneous case showed a more robust behaviour than the sequential one in terms of efficacy (AP better than AS, and NAP better than NAS) due to the better use of the total amount of resources (evaluations) available. Besides this, the simultaneous mode shows an additional advantage: it outperforms the sequential mode in terms of solving more instances in less time because the simultaneous operational mode solves first the easiest instances, leaving the hard ones to the last stages of the search. The use of the adaptation and the rewarding mechanism proposed was clearly beneficial. In particular AS was better than NAS, and AP was better than NAP, since they needed to use a lower number of evaluations to solve the same set of instances. The developed tool and the results obtained are reported in the Soft Computing journal [107].

Before finishing, we should point out that to carry out the experimentation done in this thesis we make use of two different tools developed in the Model of Decisions and Optimisation research group:

- SiGMA: an optimisation-based decision support system that provides a powerful and dynamic manager for algorithmic solvers facilitating its management and comparison as well as the analysis of their results
- DACOS: an integrated system for helping in the design and analysis of centralised cooperative optimization systems

SiGMA was used for the tuning of the different solvers that composed the strategies, whereas DACOS was employ in the configuration and set up of the cooperative methods evaluated in this thesis. The author contributed to the improvements of these tools and hence to their publication in the scientific journals Expert Systems with Applications [68] and Software: Practice and Experience [41] (submitted), respectively.

7.2 Future Work

This last section is devoted to provide the new ideas, improvement possibilities and research lines that have arisen from the work done in this

dissertation. These ideas will be associated with chapters with the aim of facilitating its understanding.

Firstly, studies done in chapter 4 can be extended in multiple ways. Regarding the composition of the strategy, other possibilities can be considered. For instance, it would be interesting to assess the behaviour of the cooperative method when a population-based algorithm is incorporated as solver or when all solvers are population-based. More research on the cooperation scheme should also be done. The behaviour of the strategy presented can be improved by designing new antecedents (condition) or consequents (action) for the control rule. For the antecedent, the utilization of techniques as Machine Learning could provide rules that adapt its working according to certain features of the current instance by means of a previous learning phase. Self-adaptation can be another methodology to take into account. The design of a mechanism that adjusts the parameters of the rules as a function of the state of the search can be an interesting aspect to study. Undoubtedly, the action part of the rule also deserves more research. In this dissertation we have seen two different ways to relocate solvers within the search space but many other possibilities are still unexplored (an intermediate point between the best global solution and the best solution of the solver, a point in an unexplored region, etc.). Until now, we have only discussed about actions in which the coordinator controlled the location of the solvers. Another alternative, that can be complementary to the last one, is the tuning of important parameters of the solvers by the coordinator. This can allow to increase/deacrease the exploitation/exploration balance of a method depending on the necessities of the whole search or make more similar the behaviour of a solver to another which is showing a better performance.

As for the application of centralised cooperative strategies to DOPs, there are still many unanswered questions. We want to study what type of cooperations responses better to the changes in the problem. Analysing the performance of the centralised cooperative strategy in other type of dynamism (constraints changes over the time, etc) is another aspect that can deserve future research. We are also interested in studying another application scenarios apart from DOPs. Continuous optimisation is another field in which as far as we know this type of methods have not been applied. Some preliminary experiments showed that a similar approach to the one

use for DOPs can obtain competitive results with respect to state-of-the-art algorithms.

From the work presented in chapter 6, several venues of research are opened. Testing the strategy over a different problem can be one of them. A drawback of SAT is that the fitness function is far from smooth. Measuring the fitness as the number of non-satisfied clauses lead to a reduced number of different fitness values, thus forcing us to take a drastic credit assignment scheme: an operator is rewarded if it solved an instance. Anyway, we should remark again that this scheme was successful for this problem. Potential candidates for testing are knapsack problems, where the type and difficulty of the instances can be easily controlled, and those problems belonging to the class of continuous optimisation. Secondly, credit assignment is a key point to address and its definition would depend on the type of problem considered. In the strategy described in section 5.2, the credit obtained by a particular operator is associated with its performance (the number of solved instances in our case). However, we consider that the computational cost implied by each operator should be taken into account. For example a linear operator and a quadratic one should not be rewarded with the same credit when they both produce a gain of Δ units. Finally, the adaptation mechanism deserves further research. An interesting possibility could be the self-adjustment method proposed in [59], but also, we are interested in including this idea of having multiple instances in the context of multimemetic algorithms, to check whether the evolutive adaptation produced from several instances leads to more robust strategies.

7.3 Publications

Derived from this thesis

International Journals and Book Chapters

- **A. D. Masegosa**, D. Pelta, I. G. del Amo, and J. L. Verdegay. On the Performance of Homogeneous and Heterogeneous Cooperative Search Strategies. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2008)*, volume 236 of *Studies in Computational Intelligence*, pages 287–300. Springer Berlin, 2009.

- **A. D. Masegosa**, F. Mascia, D. Pelta, and M. Brunato. Cooperative strategies and reactive search: A hybrid model proposal. In *Learning and Intelligent Optimization (LION 3)*, volume 5851 of *Lecture Notes in Computer Science*, pages 206–220. Springer Berlin, 2009.

- I. J. García, J. R. González, and **A. D. Masegosa**. A cooperative strategy for solving dynamic optimization problems. *Memetic Computing*, 2010. doi: 10.1007/s12293-010-0031-x. In press.

- **A. D. Masegosa**, D. A. Pelta, and J. R. González. Solving multiple instances at once: the role of search and adaptation. *Soft Computing*, 2010.doi: 10.1007/s00500-010-0564-4 In press.

- **A. D. Masegosa**, A. Sancho-Royo, and D. Pelta. An adaptive metaheuristic for the simultaneous resolution of a set of instances. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2007)*, volume 129 of *Studies in Computational Intelligence*, pages 125–137. Springer Berlin, 2008.

- J. R. González, D. A. Pelta, and **A. D. Masegosa**. A framework for developing optimization-based decision support systems. *Expert Systems With Applications*, 36(3P1):4581–4588, 2009.

- I. J. G. del Amo, D. A. Pelta, **A. D. Masegosa**, and J. L. Verdegay. A software modeling approach for the design and analysis of cooperative optimization systems. *Software: Practice and Experience*, 2009. (Submitted).

Conference Publications

- **A. D. Masegosa**, F. Mascia, D. Pelta, and M. Brunato. Control rules in cooperative strategies. In *Learning and Intelligent OptimizatioN Conference (LION 3)*, 2009. Extended abstract.

Related with this thesis (collaborator)

Book Chapters

- **A. D. Masegosa**, E. Muñoz, D. Pelta, and J. M. Cadenas. Using knowledge discovery in cooperative strategies: two case studies. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010),Studies in Computational Intelligence*. Springer Berlin, 2010. In press.

Conference Publications

- **A. D. Masegosa**, A. Sancho-Royo, and D. Pelta. Una metaheurística híbrida auto-adaptativa multi-capa para la resolución simultánea de múltiples instancias. In *I Jornadas sobre Algoritmos Evolutivos y Metaheurísticas (JAEM 2007)*, pages 57–63. 2007.

- J. R. González, **A. D. Masegosa**, I. J. García, and D. Pelta. Cooperation rules in a trajectory-based centralised cooperative strategy for dynamic optimisation problems. Submitted to IEEE International Conference on Evolutionary Computation (CEC2010). 2010.

Bibliography

[1] E. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, 1997.

[2] E. Aarts and G. Verhoeven. *Handbook of Evolutionary Computation*, chapter Genetic local search for the traveling salesman problem, pages G9.5:1–7. Institute of Physics Publishing and Oxford University Press, 1997.

[3] F. Abbattista, N. Abbattista, and L. Caponetti. An evolutionary and cooperative agent model for optimization. In *IEEE International Conference on Evolutionary Computation (CEC95)*, pages 668–671, 1995.

[4] R. M. Aiex, S. Binato, and M. G. C. Resende. Parallel grasp with path-relinking for job shop scheduling. *Parallel Computing*, 29(4):393–430, 2003.

[5] E. Alba and B. Dorronsoro. *Cellular genetic algorithms*. Springer Verlag, 2008.

[6] V. Bachelet, Z. Hafidi, P. Preux, and E.-G. Talbi. Diversifying tabu search by genetic algorithms. In *Operations Research and Management Sciences Meeting (INFORMS'98)*, 1998.

[7] T. Bäck. The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In R. Männer and B. Manderick, editors, *2nd International Conference on Parallel Problem Solving from Nature (PPSN II)*, pages 85–94, 1992.

[8] R. Battiti, M. Brunato, and F. Mascia. *Reactive search and intelligent optimization*. Springer, 2008.

[9] R. Battiti, M. Brunato, and F. Mascia. *Reactive Search and Intelligent Optimization*, volume 45 of *Operations research/Computer Science Interfaces*. Springer Verlag, 2008.

[10] R. Battiti and F. Mascia. Reactive local search for maximum clique: A new implementation. Technical Report DIT-07-018, Department of Information and Communication Technology, University of Trento, May 2007.

[11] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.

[12] J. Beasley. Obtaining test problems via internet. *Journal of Global Optimization*, 8(4):429–433, 1996.

[13] M. Birattari. *The Problem of Tuning Metaheuristics as seen from a machine learning perspective*, volume 292 of *Dissertations in Artificial Intelligence*. IOS Press, 2005.

[14] T. Blackwell and P. J. Bentley. Dynamic search with charged swarms. In *Genetic and Evolutionary Computation Conference (GECCO '02)*, pages 19–26. Morgan Kaufmann Publishers Inc., 2002.

[15] T. Blackwell and J. Branke. Multi-swarm optimization in dynamic environments. In G. R. Raidl, editor, *Applications of Evolutionary Computing*, volume 3005 of *Lecture Notes in Computer Sciences*, pages 489–500. Springer, 2004.

[16] T. Blackwell and J. Branke. Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation*, 10(4):459–472, 2006.

[17] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[18] G. Bongiovanni, P. Crescenzi, and C. Guerra. Parallel simulated annealing for shape detection. *Computer vision and image understanding*, 61(1):60–69, 1995.

[19] P. P. Bonissone. Soft computing: the convergence of emerging reasoning technologies. *Soft Computing*, 1(1):6–18, 1997.

[20] J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In *IEEE Congress on Evolutionary Computation CEC99*, pages 1875–1882. IEEE, 1999.

[21] J. Branke. *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, 2001.

[22] J. Branke and H. Schmeck. *Advances in evolutionary computing: theory and applications*, chapter Designing evolutionary algorithms for dynamic optimization problems, pages 239–262. Natural Computing. Springer-Verlag, 2003.

[23] O. Bräysy. A reactive variable neighborhood search for the vehicle-routing problem with time windows. *INFORMS Journal on Computing*, 15(4):347–368, 2003.

[24] J. Campbell, A. Ernst, and M. Krishnamoorthy. *Facility Location: Applications and Theory*, chapter Hub location problems, pages 373–406. Springer-Verlag, 2002.

[25] E. Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis*, 10(2):141–171, 1998.

[26] V. Cerny. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

[27] K. Chakhlevitch and P. Cowling. *Adaptive and Multilevel Metaheuristics*, volume 136 of *Studies in Computational Intelligence*, chapter Hyperheuristics: Recent developments, pages 3–29. Springer Berlin, 2008.

[28] H. Chen and N. S. Flann. Parallel simulated annealing and genetic algorithms: a space of hybrid methods. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *3rd International Conference on Parallel Problem Solving from Nature (PPSN III)*, volume 866 of *Lecture Notes in Computer Science*, pages 428–438. Springer Berlin, 1994.

[29] P. Chu. *A genetic algorithm approach for combinatorial optimization problems*. PhD thesis, Management School, Imperial College of Science, London, 1997.

[30] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[31] D. Corne, M. Dorigo, and F. Glover, editors. *New Ideas in Optimization*. McGraw-Hill, 1999.

[32] T. G. Crainic. *Metaheuristic Optimization Via Memory and Evolution: Tabu Search and Scatter Search*, chapter Parallel computation, cooperation, tabu search, pages 283–302. Kluwer Academic Publishers, 2005.

[33] T. G. Crainic, M. Gendreau, P. Hansen, and N. Mladenovic. Cooperative parallel variable neighborhood search for the p-median. *Journal of Heuristics*, 10(3):293–314, 2004.

[34] T. G. Crainic, A. T. Nguyen, and M. Gendreau. Cooperative multithread parallel tabu search with evolutionary adaptive memory. In *2nd International Conference on Metaheuristics (MIC'97)*, 1997.

[35] T. G. Crainic and H. Nourredine. *Parallel Metaheuristics*, chapter Parallel meta-heuristics applications, pages 447–494. John Wiley & Sons, 2005.

[36] T. G. Crainic and M. Toulouse. *Fleet Management and Logistics*, chapter Parallel Metaheuristics. Kluwer Academic, 1998.

[37] T. G. Crainic and M. Toulouse. Explicit and emergent cooperation schemes for search algorithms. In *Learning and Intelligent Optimization: Second International Conference (LION II)*, volume 5313 of *Lecture Notes in Computer Science*, pages 95–109, 2008.

[38] T. G. Crainic, M. Toulouse, and M. Gendreau. Synchronous tabu search parallelization strategies for multicommodity location-allocation with balancing requirements. *OR Spectrum*, 17(2-3):113–123, 1995.

[39] T. G. Crainic, M. Toulouse, and M. Gendreau. Toward a taxonomy of parallel tabu search heuristics. *INFORMS Journal on Computing*, 9(1):61–72, 1997.

[40] C. Cruz and D. Pelta. Soft computing and cooperative strategies for optimization. *Applied Soft Computing*, 9(1):30–38, 2009.

[41] I. J. G. del Amo, D. A. Pelta, A. D. Masegosa, and J. L. Verdegay. A software modeling approach for the design and analysis of cooperative optimization systems. *Submited to Software: Practice and Experience*, 2009.

[42] J. L. Deneubourg, S. Aron, S. Goss, and J. M. Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior*, 3(2):159–168, 1990.

[43] M. Dorigo, M. Birattari, and T. Stützle. Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39, 2006.

[44] M. Dorigo and G. D. Caro. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, 1999.

[45] M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1):29–41, 1996.

[46] M. Dorigo and T. Stützle. *Ant Colony Optimization*. Bradford Books. The MIT Press, 2004.

[47] R. Eberhart and J. Kennedy, editors. *Swarm Intelligence*. Academic Press, 2001.

[48] R. C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *IEEE International Congress on Evolutionary Computation (CEC2000)*, pages 84–88, 2000.

[49] A. E. Eiben, P.-E. Raué, and Z. Ruttkay. Genetic algorithms with multi-parent recombination. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *3rd International Conference on Parallel Problem Solving from Nature (PPSN III)*, volume 866 of *Lecture Notes in Computer Sciences*, pages 78–87. Springer Berlin, 1994.

[50] A. E. Eiben and Z. Ruttkay. *Handbook of Evolutionary Computation*, chapter Constraint-satisfaction problems, pages C5.7:1–8. Institute of Physics Publishing and Oxford University Press, 1997.

[51] T. A. Feo, M. G. C. Resende, and S. H. Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, pages 860–878, 1994.

[52] J. Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[53] M. A. Fleischer. Cybernetic optimization by simulated annealing: Accelerating convergence by parallel processing and probabilistic feedback control. *Journal of Heuristics*, 1(2):225–246, 1996.

[54] C. Fluerent and J. A. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63:437–461, 1995.

[55] T. Fogarty and R. Huang. Implementing the genetic algorithm on transputer based parallel processing systems. In *1st International Conference on Parallel Problem Solving from Nature (PPSN I)*, volume 496 of *Lecture Notes in Computer Sciences*, pages 145–149. Springer Berlin, 1991.

[56] L. Fogel, A. Owens, M. Walsh, and J. Lawrence. *Artificial Intelligence Through Simulated Evolution*. Wiley, 1966.

[57] L. J. Fogel. Toward inductive inference automata. In *International Federation for Information Processing Congress*, pages 395–399, 1962.

[58] F. Franzè and N. Speciale. A tabu-search-based algorithm for continuous multiminima problems. *International Journal for numerical methods in engineering*, 50(3):665–680, 2001.

[59] M. Gagliolo and J. Schmidhuber. Learning dynamic algorithm portfolios. *Annals of Mathematics and Artificial Intelligence*, 47(3-4):295–328, 2006.

[60] I. J. García, J. R. González, and A. D. Masegosa. A cooperative strategy for solving dynamic optimization problems. *Memetic Computing*, 2010. DOI: 10.1007/s12293-010-0031-x. In press.

[61] I. Gent, H. H. Hoos, P. Prosser, and T. Walsh. Morphing: Combining structure and randomness. In *6th National Conference on Artificial Intelligence (AAAI'99)*, pages 654–660, 1999.

[62] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

[63] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[64] F. Glover, M. Laguna, and R. Marti. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39(3):653–684, 2000.

[65] F. Glover and B. Melián. Búsqueda tabú. *Revista Iberoamericana de Inteligencia Artificial*, 7(19):29–48, 2003.

[66] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[67] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *2nd International Conference on Genetic Algorithms and their application*, pages 41–49. Lawrence Erlbaum Associates, Inc., 1987.

[68] J. R. González, D. A. Pelta, and A. D. Masegosa. A framework for developing optimization-based decision support systems. *Expert Systems With Applications*, 36(3P1):4581–4588, 2009.

[69] S. Goss, S. Aron, J. Deneubourg, and J. Pasteels. Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, 76(12):579–581, 1989.

[70] P.-P. Grassé. La reconstruction du nid et les coordinations interindividuelles chezbellicositermes natalensis etcubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6(1):41–80, 1959.

[71] J. Grefenstette. Incorporating problem specific knowledge into genetic algorithms. In L. Davis, editor, *Genetic algorithms and simulated annealing*, pages 42–60, 1987.

[72] P. Hansen and N. Mladenovic. Variable neighborhood search: principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001.

[73] P. Hansen, N. Mladenovic, and J. A. Moreno-Pérez. Búsqueda de entorno variable. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, 19:77–92, 2003.

[74] G. R. Harik, F. G. Lobo, and K. Sastry. *Scalable Optimization via Probabilistic Modeling*, volume 33 of *Studies in Computational Intelligence*, chapter Linkage Learning via Probabilistic Modeling in the Extended Compact Genetic Algorithm (ECGA), pages 39–61. Springer Berlin, 2007.

[75] W. Hart, N. Krasnogor, and J. Smith, editors. *Recent Advances in Memetic Algorithms*. Studies in Fuzziness and Soft Computing. Physica-Verlag, 2004.

[76] A. Hedar and M. Fukushima. Heuristic pattern search and its hybridization with simulated annealing for nonlinear global optimization. *Optimization Methods and Software*, 19(3-4):291–308, 2004.

[77] A.-R. Hedar and M. Fukushima. Tabu search directed by direct search methods for nonlinear global optimization. *European Journal of Operational Research*, 170(2):329–349, 2006.

[78] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[79] H. Hoos and T. Stützle. Local search algorithms for SAT: An empirical evaluation. *Journal of Automated Reasoning*, 24(4):421–481, 2000.

[80] H. Hoos and T. Stützle. SATLIB: An online resource for research on SAT. In H. van Maaren, I. P. Gent, and T. Walsh, editors, *SAT2000*, pages 283–292. IOS Press, 2000.

[81] E. Horvitz, Y. Ruan, C. P. Gomes, H. A. Kautz, B. Selman, and D. M. Chickering. A bayesian approach to tackling hard computational problems. In *7th Conference in Uncertainty in Artificial Intelligence (UAI '01)*, pages 235–244. Morgan Kaufmann, 2001.

[82] B. A. Huberman, R. M. Lukose, and T. Hogg. An economics approach to hard computational problems. *Science*, 275(5296):51–54, 1997.

[83] P. Husbands, F. Mill, and S. Warrington. Genetic algorithms, production plan optimisation, and scheduling. In *1st International Conference on Parallel Problem Solving from Nature (PPSN I)*, volume 496 of *Lecture Notes in Computer Science*, pages 80–84. Springer-Verlag, 1991.

[84] F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In *22nd National Conference on Artifical Intelligence (AAAI'07)*, pages 1152–1157, 2007.

[85] T. James, C. Rego, and F. Glover. A cooperative parallel tabu search algorithm for the quadratic assignment problem. *European Journal of Operational Research*, 195(3):810–826, 2009.

[86] P. Jog, J. Y. Suh, and D. van Gucht. The effects of population size, heuristic crossover and local improvement on a genetic algorithm for the traveling salesman problem. In *3rd International Conference on Genetic algorithms*, pages 110–115. Morgan Kaufmann Publishers Inc., 1989.

[87] J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[88] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[89] J. Kratica, Z. Stanimirović, and V. F. Dušcan Tovšić. Two genetic algorithms for solving the uncapacitated single allocation p-hub median problem. *European Journal of Operational Research*, 182(1):15–28, 2007.

[90] M. G. Lagoudakis and M. L. Littman. Algorithm selection using reinforcement learning. In *7th International Conference on Machine Learning (ICML '00)*, pages 511–518. Morgan Kaufmann, 2000.

[91] X. Li. Efficient differential evolution using speciation for multimodal function optimization. In *Conference on Genetic and evolutionary computation (GECCO '05)*, pages 873–880. ACM, 2005.

[92] X. Li, D. Ruan, and A. J. van der Wal. Discussion on soft computing at FLINS'96. *International Journal of Intelligent Systems*, 13(2-3):287–300, 1998.

[93] F.-T. Lin, C.-Y. Kao, and C.-C. Hsu. Incorporating genetic algorithms into simulated annealing. In F. J. Cantu-Ortiz and H. Terashima-Marin, editors, *International Symposium on Artificial Intelligence*, pages 290–297. LIMUSA, 1991.

[94] L. Lin and G. Mitsuo. Auto-tuning strategy for evolutionary algorithms: balancing between exploration and exploitation. *Soft Computing*, 13(2):157–168, 2008.

[95] F. G. Lobo, C. F. Lima, and Z. Michalewicz. *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*. Springer, 2007.

[96] R. Love, J. Moris, and G. Wesolowsky. *Facility location: Models and methods*. 1988.

[97] M. Lozano, F. Herrera, N. Krasnogor, and D. Molina. Real-coded memetic algorithms with crossover hill-climbing. *Evolutionary Computation*, 12(3):273–302, 2004.

[98] S. W. Mahfoud. Crowding and preselection revisited. In R. Männer and B. Manderick, editors, *2nd International Conference on Parallel Problem Solving from Nature (PPSN II)*, pages 27–36, 1992.

[99] S. W. Mahfoud and D. E. Goldberg. Parallel recombinative simulated annealing: a genetic algorithm. *Parallel Computing*, 21(1):1–28, 1995.

[100] C. E. Mariano and E. Morales. A multiple objective ant-q algorithm for the design of water distribution irrigation networks. In *1st International Workshop on Ant Colony Optimization (ANTS'98)*, 1998.

[101] R. Martí. *Handbook of Metaheuristics*, chapter Multistart methods, pages 355–368. Kluwer Academic, 2003.

[102] R. Martí and J. M. Moreno. Métodos multiarranque. *Revista Iberoamericana de Inteligencia Artificial*, 19:49–60, 2003.

[103] O. Martin, S. Otto, and E. Felten. Large-step markov chains for the TSP incorporating local search heuristics. *Operations Research Letters*, 11(4):219–224, 1992.

[104] A. D. Masegosa, F. Mascia, D. Pelta, and M. Brunato. Control rules in cooperative strategies. In *Learning and Intelligent OptimizatioN Conference (LION 3)*, 2009. Extended abstract.

[105] A. D. Masegosa, F. Mascia, D. Pelta, and M. Brunato. Cooperative strategies and reactive search: A hybrid model proposal. In *Learning and Intelligent Optimization (LION 3)*, volume 5851 of *Lecture Notes in Computer Science*, pages 206–220. Springer Berlin, 2009.

[106] A. D. Masegosa, D. Pelta, I. G. del Amo, and J. L. Verdegay. On the Performance of Homogeneous and Heterogeneous Cooperative Search Strategies. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2008)*, volume 236 of *Studies in Computational Intelligence*, pages 287–300. Springer Berlin, 2009.

[107] A. D. Masegosa, D. A. Pelta, and J. R. González. Solving multiple instances at once: the role of search and adaptation. *Soft Computing*, 2010. DOI: 10.1007/s00500-010-0564-4. In press.

[108] J. Mateo and L. de la Ossa. LiO: Tool for metaheuristics. *http://www.info-ab.uclm.es/simd/SOFTWARE/LIO/*, 2006.

[109] B. Mazure, L. Sais, and E. Grégoire. Tabu search for SAT. In *14th National Conference on Artificial Intelligence (AAAI'97)*, pages 281–285, 1997.

[110] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *14th National Conference on Artificial Intelligence (AAAI'97)*, pages 321–326, 1997.

[111] M. Mehdi, N. Melab, E.-G. Talbi, and P. Bouvry. Interval island model initialization for permutation-based problems. In *11th Annual conference on Genetic and Evolutionary Computation (GECCO'09)*, pages 1919–1920. ACM, 2009.

[112] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1998.

[113] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[114] P. Moscato. Memetic algorithms: a short introduction. In D. Corne, M. Dorigo, and F. Glover, editors, *New ideas in optimization*, pages 219–234. McGraw-Hill Ltd., UK, 1999.

[115] H. Mühlenbein and H.-M. Voigt. Gene pool recombination in genetic algorithms. In I. Osman and J. Kelly, editors, *Metaheuristics International Conference (MIC '96)*, pages 53–62, 1996.

[116] M. O'Kelly and E. Morton. A quadratic integer program for the location of interacting hub facilities. *European Journal of Operational Research*, 32(3):393–404, 1987.

[117] Y.-S. Ong, M.-H. Lim, N. Zhu, and K.-W. Wong. Classification of adaptive memetic algorithms:a comparative study. *IEEE Transactions On Systems, Man, and Cybernetics-Part B: Cybernetics*, 36(1):141–152, 2006.

[118] U.-M. O'Reilly and F. Oppacher. Hybridized crossover-based search techniques for program discovery. In *IEEE International Conference on Evolutionary Computation (CEC95)*, volume 2, pages 573–578. IEEE Press, 1995.

[119] I. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63(5):511–623, 1996.

[120] J. M. Pasteels, J. L. Deneubourg, and S. Goss. Self-organization mechanisms in ant societies (I): trail recruitment to newly discovered food sources. *Experientia Supplementum*, 54:155–175, 1987.

[121] D. Pelta, A. Blanco, and J. Verdegay. A fuzzy valuation-based local search framework for combinatorial optimization problems. *Journal of Fuzzy Optimization and Decision Making*, 1(2):177–193, 2002.

[122] D. Pelta, C. Cruz, and J. R. González. A study on diversity and cooperation in a multiagent strategy for dynamic optimization problems. *International Journal of Intelligent Systems*, 24(7):844–861, 2009.

[123] D. Pelta, C. Cruz, and J. L. Verdegay. Simple control rules in a cooperative system for dynamic optimisation problems. *International Journal of General Systems*, 38(7):701–717, 2008.

[124] D. Pelta and N. Krasnogor. Multimeme algorithms using fuzzy logic based memes for protein structure prediction. In W. Hart, N. Krasnogor, and J. Smith, editors, *Recent Advances in Memetic Algorithms*, volume 166 of *Studies in Fuzziness and Soft Computing*, pages 49–54. Physica-Verlag, 2004.

[125] D. Pelta, A. Sancho-Royo, C. Cruz, and J. Verdegay. Using memory and fuzzy rules in a co-operative multi-thread strategy for optimization. *Information Sciences*, 176(13):1849–1868, 2006.

[126] M. Petrik and S. Zilberstein. Learning parallel portfolios of algorithms. *Annals of Mathematics and Artificial Intelligence*, 48(1-2):85–106, 2006.

[127] G. R. Raidl. A unified view on hybrid metaheuristics. In F. Almeida, M. J. B. Aguilera, C. Blum, J. M. Moreno-Vega, M. P. Pérez, A. Roli, and M. Sampels, editors, *3rd International Workshop on Hybrid Metaheuristics*, volume 4030 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin, 2006.

[128] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog, 1973.

[129] C. R. Reeves and J. E. Rowe. *Genetic algorithms: principles and perspectives: a guide to GA theory.* Kluwer Academic Publishers, 2003.

[130] J. Renders and H. Bersini. Hybridizing genetic algorithms with hill-climbing methods for global optimization: two possible ways. In *1st IEEE International Conference on Evolutionary Computation (CEC94)*, pages 312–317, 1994.

[131] J. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[132] H. Richter and S. Yang. Learning behavior in abstract memory schemes for dynamic optimization problems. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 13(12):1163–1173, 2009.

[133] D. Schuurmans and F. Southey. Local search characteristics of incomplete SAT procedures. *Artificial Intelligence*, 132(2):121–150, 2001.

[134] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *12th National Conference on Artificial Intelligence (AAAI'95)*, pages 337 – 343. MIT Press, 1994.

[135] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *9th National Conference on Artificial Intelligence (AAAI'92)*, pages 440–446, 1992.

[136] K. Shahookar and P. Mazumder. A genetic approach to standard cell placement using metagenetic parameter optimization. *IEEE Transactions on Computer-Aided Design*, 9(5):500–511, 1990.

[137] J. Singer, I. P. Gent, and A. Smaill. Backbone fragility and the local search cost peak. *Journal of Artificial Intelligence Research*, 12(1):235–270, 2000.

[138] J. Smith. Coevolving memetic algorithms: A review and progress report. *IEEE Transactions On Systems, Man, And Cybernetics-Part B: Cybernetics*, 37(1):6–17, 2007.

[139] J. Smith. Credit assignment in adaptive memetic algorithms. In *Genetic and Evolutionary Computation Conference 2007 (GECCO2007)*, pages 1412–1419, 2007.

[140] J. Smith and T. Fogarty. Operator and parameter adaptation in genetic algorithms. *Soft Computing*, 1(2):81–87, 1997.

[141] T. Stützle and H. Hoos. The MAX-MIN ant system and local search for combinatorial optimization problems: Toward adaptive tools for global optimisation. In *2nd International Conference on Metaheuristics (MIC'97)*, pages 191–193, 1997.

[142] G. Syswerda. Simulated crossover in genetic algorithms. In L. D. Whitley, editor, *2nd Workshop on Foundations of Genetic Algorithms*, pages 239–255. Morgan-Kaufmann, 1993.

[143] E. Talbi. A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8(5):541–564, 2002.

[144] J. Thiel and S. Voss. Some experiences on solving multiconstraint zero-one knapsack problems with genetic algorithms. *INFOR-Information Systems and Operational Research*, 32(4):226–242, 1994.

[145] M. Toulouse, T. G. Crainic, and B. Sanso. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, chapter An Experimental Study of The Systemic Behavior of Cooperative Search Algorithms, pages 373–392. Kluwer Academic Publishers, 1999.

[146] M. Toulouse, T. G. Crainic, B. Sanso, and K. Thulasiraman. Self-organization in cooperative tabu search algorithms. In *1998 IEEE International Conference on Systems, Man, and Cybernetics*, pages 2379–2385. Omnipress, 1998.

[147] J. L. Verdegay, R. R. Yager, and P. P. Bonissone. On heuristics as a fundamental constituent of soft computing. *Fuzzy Sets and Systems*, 159:846–855, 2008.

[148] M. Vose. *The Simple Genetic Algorithm: Foundations and Theory.* Bradford Books, 1999.

[149] S. Voss. *Network Optimization Problems*, chapter Tabu Search: Applications and Prospects, pages 333–353. World Scientific, 1993.

[150] S. Voss, S. Martello, I. H. Osman, and C. Roucairol, editors. *Meta-heuristics : advances and trends in local search paradigms for optimization.* Kluwer Academic Publishers, Boston, 1999.

[151] R. A. Watson, G. Hornby, and J. B. Pollack. Modeling building-block interdependency. In *5th International Conference on Parallel Problem Solving from Nature (PPSN V)*, volume 1498 of *Lecture Notes In Computer Science*, pages 97–108. Springer-Verlag, 1998.

[152] T. White and F. Oppacher. Adaptive crossover using automata. In *3rd International Conference on Parallel Problem Solving from Nature (PPSN III)*, volume 866 of *Lecture Notes in Computer Science*, pages 229–238, 1994.

[153] Y.-Y. Wong, K.-H. Lee, K.-S. Leung, and C.-W. Ho. A novel approach in parameter adaptation and diversity maintenance for genetic algorithms. *Soft Computing*, 7(8):506–515, 2003.

[154] A. H. Wright. Genetic algorithms for real parameter optimization. In *Foundations of Genetic Algorithms*, pages 205–218. Morgan Kaufmann, 1991.

[155] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.

[156] S. Yussof, R. A. Razali, O. H. See, A. A. Ghapar, and M. M. Din. A coarse-grained parallel genetic algorithm with migration for shortest path routing problem. In *10th IEEE International Conference on High Performance Computing and Communications*, pages 615–621. IEEE Computer Society, 2009.

[157] L. Zadeh. Fuzzy sets. *Information Control*, 8:338–353, 1965.

[158] L. Zadeh. Fuzzy logic and soft computing: issues, contentions and perspectives. In *3rd International Conference on Fuzzy Logic,Neural Nets and Soft Computing (IIZUKA'94)*, pages 1–2, 1994.

[159] L. A. Zadeh. Applied soft computing - foreword. *Applied Soft Computing*, 1(1):1–2, 2001.