University of Almería
Department of Informatics

Ph.D. Thesis

High performance computing applied to competitive facility location and design problems: single and multi-objective optimization algorithms

Aránzazu Gila Arrondo
Almería, March 2013

Ph.D. Thesis

High performance computing applied to competitive facility location and design problems: single and multi-objective optimization algorithms

University of Almería
Department of Informatics

Author: Aránzazu Gila Arrondo
Supervisors: Jose Fernández Hernández
Pilar Martínez Ortigosa
Juana López Redondo

Almería, March 2013

A mis padres, a mi hermano
y a Manu

Agradecimientos

En primer lugar me gustaría agradecer a mis tres supervisores, Dra. Dª Pilar Martínez Ortigosa, Dr. D. José Fernánez Hernández y Dra. Dª Juana López Redondo, su confianza y su dedicación. Creo sinceramente que no podría haber tenido mejores directores. A Pilar, por haber confiado en mí desde el principio y hacer que, lo que iba a ser sólo un Proyecto Final de Carrera, se convierta hoy en una Tesis Doctoral. A Jose, por darme la oportunidad de trabajar con él y hacer estos tres años de trabajo más llevaderos. A Juani, por todas las horas que me ha dedicado, sin importar las diferencias horarias desde cualquier parte del mundo, y sobre todo, por hacerme ver el lado positivo que me animaba a seguir. Y a todos ellos agradecerles, que por sus conocimientos, ha sido posible que yo haya desarrollado esta tesis.

Todo este trabajo no lo hubiese podido realizar sin financiación. Por ello, me gustaría agradecer especialmente a la Fundación Séneca, Agencia de Ciencia y Tecnología de la Región de Murcia, la concesión de la beca asociada al Proyecto 'Aplicación de la supercomputación a problemas matemáticos de optimización, algebraicos y de modelización (00003/CS/10)'. Agradezco en especial a Dr. D. Francisco Esquembre, investigador principal del citado proyecto, su constante preocupación por el bienestar de los becarios.

Notable reconocimiento merece el Centro de Supercomputación del Parque Científico de Murcia, sin el cual la mayoría de los estudios computacionales de esta tesis no se hubieran podido realizar. En particular, me gustaría destacar al servicio técnico de dicho centro, por su ayuda y colaboración.

Al grupo de investigación 'Supercomputación: Algoritmos' de la Universidad de Almería por esos buenos momentos que han hecho que el trabajo sea más ameno y entretenido. En especial a Dra Dª Inmaculada García Fernández, quien me abrió las puertas a la investigación y ha seguido mis pasos de cerca.

A Dra. Dª Boglárka G.-Tóth, de la Budapest University of Technology an Economics, por haberme permitido usar los códigos de los algoritmos de ramificación y acotación basados en el análisis de intervalos que ella implementó.

Muy importante para el desarrollo de esta tesis han sido mis estancias en el extranjero, que me han aportado nuevos puntos de vista en mi investigación. Por ello, me

gustaría citar al 'Edinburgh Parallel Computing Centre (EPCC)' y al 'Swiss National Supercomputing Center (CSCS)', por haberme brindado la posibilidad de utilizar sus recursos tanto técnicos como científicos. En particular, me gustaría agradecer a D. Michele De Lorenzi y Dr. D. William Sawyer su acogida y colaboración.

A Manu por su apoyo, comprensión, paciencia y ánimo en los momentos de desaliento. Por haber creído en mí siempre y, porque a pesar de la distancia, ha estado a mi lado constantemente.

A mis padres por ser ejemplo de esfuerzo y perseverancia, y por su entera dedicación y motivación.

A mi hermano Miguel por involucrarse en todos mis proyectos, ayudándome de una manera muy especial con los pequeños y grandes problemas que se me han presentado, siendo siempre mi fuente de conocimiento. Y a Ka Lo Chan por su comprensión.

A Paco, Rosa y Clara por hacerme más llevaderos estos años de trabajo y hacerme sentir permanentemente su cariño y apoyo.

A Mariquilla y a Pachús por estar siempre animándome y empujándome para que no decayera.

A mis amigos por entender mis ausencias y mis ratos de mal humor, y por haberme demostrado que son eso, muy buenos amigos.

Desde luego, esta tesis no hubiese sido posible sin todas y cada una de las personas e instituciones aquí citadas. Muchas gracias a todos.

Prefacio

La localización de servicios pretende encontrar el emplazamiento de uno o más centros (servicios) de modo que se optimice una o varias funciones objetivo. Dicha función objetivo puede, por ejemplo, tratar de minimizar el coste de transporte, proporcionar a los clientes un servicio de forma equitativa, capturar la mayor cuota de mercado posible, etc. La localización de servicios abarca muchos campos, como la investigación operativa, la ingeniería industrial, la geografía, la economía, las matemáticas, el marketing, el planning urbanístico, además de otros muchos campos relacionados.

Existen muchos problemas de localización en la vida real, como por ejemplo, la localización de hospitales, de colegios o vertederos, por nombrar algunos. Para ser capaces de obtener soluciones a los problemas de localización, es necesario desarrollar/diseñar un modelo que represente la realidad lo más fielmente posible. Dichos modelos pueden llegar a ser realmente difíciles de tratar. Para resolver tales problemas de localización, se han propuesto muchos *algoritmos de optimización global*, tanto *exactos* como *heurísticos*. Los algoritmos exactos se caracterizan por ser capaces de obtener el óptimo global con una cierta precisión. Los algoritmos heurísticos, sin embargo, no pueden demostrar matemáticamente su convergencia al óptimo global, aunque pueden llegar a obtener soluciones tan buenas y precisas como las proporcionadas por los exactos. Los exactos son altamente costosos desde el punto de vista computacional, lo que implica que, en determinados casos, sea imposible aplicarlos para resolver un problema utilizando incluso los computadores más avanzados. De este modo los algoritmos heurísticos, con menos requerimientos computacionales que los exactos, se alzan entonces como una buena alternativa. No obstante, en determinadas circunstancias, las necesidades computacionales son tan elevadas, que el uso de algoritmos heurísticos ejecutándose en procesadores estándares actuales no es suficiente. En tales situaciones, se hace necesario el uso de *computación de altas prestaciones*.

Esta tesis, "*High performance computing applied to competitive facility location and design problems: single and multi-objective optimization algorithms*" (Aplicación de la computación de altas prestaciones a problemas de localización competitiva con decisiones en diseño: algoritmos de optimización uni y multi-objetivo), proporciona, por un lado, algoritmos heurísticos capaces de resolver problemas de localización competitiva,

tanto para funciones escalares como vectoriales, y por otro lado, técnicas paralelas que permiten reducir el tiempo de ejecución, resolver problemas más grandes, e incluso, en ocasiones, mejorar la calidad de las soluciones.

Esta tesis está organizada en cuatro capítulos. El primero de ellos es una introducción a las tres áreas de investigación en las que se enmarca esta tesis: localización, algoritmos de búsqueda y computación de altas prestaciones. Primeramente se describen de forma sucinta los principales elementos de un problema de localización. A continuación, se introducen los campos de la optimización global y multi-objetivo, y se definen los correspondientes conceptos de solución. Finalmente, se revisan brevemente las arquitecturas paralelas así como los modelos de programación utilizados en esta tesis.

En el Capítulo 2, se presenta un nuevo modelo de localización competitiva en el plano de un solo centro, en el que la demanda varía dependiendo de la localización de dicho centro. Este nuevo modelo se compara con su correspondiente modelo de demanda fija, mediante algoritmos disponibles en la literatura. En particular, el algoritmo evolutivo UEGO se ha adaptado al nuevo modelo, mediante el diseño y desarrollo de un nuevo optimizador local. Además, se ha llevado a cabo un extenso estudio computacional con la intención de estudiar el impacto que tiene la consideración de demanda fija o demanda variable en la localización. Finalmente, se presenta una paralelización del algoritmo UEGO, que permite resolver problemas más costosos.

El Capítulo 3 está dedicado al problema de líder-seguidor con demanda variable. Este problema se puede considerar una extensión del modelo anterior para el caso en el que el competidor (el seguidor) reacciona localizando un nuevo centro después de que la cadena (el líder) localice su propio centro. El objetivo del líder es encontrar la solución que maximice su beneficio, teniendo en cuenta la futura reacción del seguidor. Por tanto, hay que resolver dos problemas: el problema del seguidor, también denominado *medianoide*, y el problema del líder o *centroide*. El modelo matemático se detalla al comienzo del capítulo. Posteriormente, se proponen y evalúan varios algoritmos para resolver el problema del *centroide*. TLUEGO, basado en UEGO, es el algoritmo que proporciona mejores resultados. El capítulo finaliza presentando tres paralelizaciones de TLUEGO, que permiten resolver problemas más grandes de forma precisa.

El último capítulo aborda un problema de localización bi-objetivo del franquiciado y del franquiciado. Este problema se define en el plano y ya estaba propuesto en la

literatura. Para la obtención de una buena aproximación de todo el frente de Pareto, se propone un nuevo algoritmo evolutivo, llamado FEMOEA, que también se puede emplear para resolver otros problemas multi-objetivo generales tipo caja negra en donde no se conocen las características matemáticas de tales problemas. Para la resolución del problema bi-objetivo, FEMOEA incluye una búsqueda local, basada en el gradiente, para mejorar la calidad (eficiencia) del las soluciones. FEMOEA además, presenta un criterio de finalización que permite al algoritmo parar tan pronto como obtenga una buena aproximación del frente de Pareto. Para analizar el comportamiento y buen rendimiento de FEMOEA, se ha realizado un extenso estudio computacional. Finalmente, se propone una versión paralela de FEMOEA eficiente y eficaz que permite resolver problemas de mayor dimensión.

La tesis finaliza con un resumen de los principales resultados obtenidos y apuntando líneas de investigación futura.

Preface

Facility location applications are concerned with the location of one or more facilities in a way that optimizes one or several objectives at the same time, such as minimizing transportation costs, providing equitable service to customers, capturing the largest market share, etc. The research on facility location problems spans many research fields such as operations research/management science, industrial engineering, geography, economics, computer science, mathematics, marketing, urban planning, and the corresponding related fields.

There exist many facility location problems in real life, for example location of hospitals, schools or rubbish dumps, to name a few. Nevertheless, to be able to obtain certain solutions to location problems, it is necessary to describe them as close to reality as possible, which may result in hard-to-solve optimization models. Many *global optimization algorithms*, either *exact* or *heuristic*, have flourished to cope with location problems. However, exact algorithms are unable to solve some problems, because they are too slow or the computer runs out of memory. Thus, heuristic procedures have to be developed to solve those other difficult problems. Nevertheless, sometimes, computational requirements are so high that the use of heuristic algorithms running in standard computers is not enough. In such situations, *high performance computing* is needed.

This thesis, "*High performance computing applied to competitive facility location and design problems: single and multi-objective optimization algorithms*", provides, on the one hand, heuristic algorithms able to solve single and multi-objective competitive continuous location problems and, on the other hand, parallel techniques which reduce the execution time, allow to solve larger problems and, in some cases, improve the quality of the solutions.

This thesis is organized in four chapters. The first one is an introduction to the three main research areas involved in this thesis, i.e., location science, search algorithms and high performance computing. Chapter 1 begins with a brief description of the main elements of a location problem. Then, the definition of global optimization and nonlinear multi-objective optimization problems is given, and some definitions and quality

measures are also introduced. Finally, parallel architectures and parallel programing models applied in this thesis are briefly described.

In Chapter 2, a new single facility location and design model in which 'the demand varies depending on the location of the facility' is presented, and compared to the corresponding fixed demand model. An evolutionary algorithm called UEGO proposed in literature to accurately solve the fixed demand model is adapted to the new model. In particular, a new local optimizer has been developed. Several computational studies are carried out in order to investigate to what extent the assumption of fixed demand, commonly employed in competitive location literature, has an impact on the location decision. Moreover, a sensitivity analysis is conducted in order to check the changes in optimal design/location when the model environment and parameters change. Finally, a parallelization of UEGO is presented, in order to handle the drawbacks when the complexity of the problem to be solved increases.

Chapter 3 is dedicated to a leader-follower problem with endogenous demand. It is the extension of the previous model to the case in which the competitor's chain (the follower) reacts by also locating a new facility after the locating chain (the leader) locates its own facility. The objective of the leader is to find the solution which maximizes its profit, following the location of the facility of the follower. This means that two problems are considered at the same time: the follower (or *medianoid*) problem, and the leader (or *centroid*) problem. The model of the leader-follower problem is introduced at the beginning of the chapter. Then, several algorithms to solve the *centroid* problem are proposed and evaluated. TLUEGO, based on UEGO, is the algorithm which provides better solutions. The chapter ends with three parallelizations of TLUEGO, which allow to solve larger instances accurately.

The last chapter, Chapter 4, deals with a bi-objective planar franchisor-franchisee facility location and design problem, already proposed in literature. A new memetic bi-objective evolutionary algorithm, called FEMOEA, is proposed to solve this continuous bi-objective optimization problem. In its framework, FEMOEA includes a local search, which uses gradient information, to improve the quality (efficiency) of the points, as well as a termination rule to stop the algorithm as soon as a good approximation of the Pareto-front is obtained. In order to show the behavior and good performance of FEMOEA a comprehensive computational study is carried out. Finally, an efficient and effective parallel version of FEMOEA capable of solving bigger problems is presented

at the end of the chapter.

The thesis concludes with a summary where the main results are outlined and future lines of research are presented.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

This chapter deals with the three main research areas involved in this thesis, i.e., location science, search algorithms and high performance computing. A section is devoted to each one of them, where the main issues related to the current work are described and defined briefly.

1.1 Location science

Location science deals with the location of one or more facilities in a way that optimizes a certain objective (minimization of transportation costs, minimization of social costs, maximization of the market share, etc.). For an introduction to the topic see [49, 50, 71]. All location problems share several components, which leads to different models. The mathematical formulations and methods used to solve the problems vary substantially depending on the type of model. This section analyzes the main elements of a location model.

The space where a location problem is explicitly defined determines the set of methods that can be applied. It is possible to distinguish three different location spaces: *continuous* location problems, where the facilities to be sited can be placed anywhere on a region of the plane, *network* location problems, where any point on a network is suitable for location, and *discrete* location problems, where the facilities can be located only at a limited number of eligible points. Furthermore, the feasible set may be restricted by the introduction of "forbidden zones", i.e. areas in which facilities should not be located [55] or other types of constraints. From the optimization point of view, the techniques used to cope with the problems also differ. Continuous location problems are, in most of cases, nonlinear optimization problems, while discrete and network location problems are integer programming/combinatorial optimization problems.

It is customary to differentiate between single-facility location problems and multi-

facilities problems. In the former, only one facility is to be located ($nfc = 1$), while in the latter, several facilities are to be sited ($nfc > 1$). Notice that when locating multiple new facilities, interactions between them may have to be considered, usually through the interdistances.

Two location models can be distinguished depending on whether a single player or multiple players in the market are considered. They are referred to as *non-competitive* and *competitive* models, respectively. A detailed taxonomy can be found in survey papers [57, 84, 143]. In many location models it is assumed that the decision maker, who plans the location of his facilities, faces an empty space without any similar or competing facilities. Nevertheless, in most cases, similar facilities already exist in the region and the task is to add new ones in an optimal way [57, 103, 128]. The existing facilities may belong to the decision maker's own chain or to a competitor's chain [55]. When a competition takes place, it may be *static*, which means that the competitors are already in the market, the owner of the new facility knows their characteristics and no reaction is expected from them, or *with foresight*, in which the competitors are assumed to react after the new facility enters. Furthermore, if the competitors can change their decisions, the model is considered *dynamic*, as it is characterized by the major concern of the existence of *equilibria*. In this context, Hakimi [82] introduced the well known Stackelberg problems (also known as Simpson's problems in voting theory). The scenario considered in these kind of problems is that of a *duopoly*. A chain (the leader) wants to set up *nfc* new facilities in the market, where similar facilities of a competitor (the follower), and possibly of its own chain, are already present. The follower will react by locating *nff* facilities after the leader locates its own facilities. Hakimi introduced the terms *medianoid* for the follower problem, and *centroid* for the leader problem. More precisely, an $(nff|LOC_{nfc})$ medianoid problem refers to the follower's problem of locating *nff* new facilities in the presence of *nfc* leader's facilities located at a set of points LOC_{nfc}. And an $(nff|nfc)$ centroid problem refers to the leader's problem of locating *nfc* new facilities, knowing that the follower will react positioning *nff* new facilities by solving an $(nff|LOC_{nfc})$ medianoid problem.

Regarding customers, clients can be either distributed according to some distribution function over a given set, or located at specific points (named *demand points*, see [70]) in the plane or at vertices in a network, which is the common approach in literature.

Another important feature is the so called *demand*. Demand can be either fixed (*exogenous*) or may vary (*endogenous*). In the first case, the demand is known with certainty. This is usually the case when the goods are *essential* to the customers, and then, they will buy the goods independently of the distance to the facility or the price. In the second case, goods are *inessential* to the customers, then, demand can vary depending on prices, distances to the facilities, etc.

Another important question to take into account is whether customers are free to choose the facility from which they are served. If so, knowing how customers buy goods among the existing facilities helps to estimate the market share captured by each facility [153]. The patronizing behavior of the customers is usually assumed to be either *deterministic*, when the full demand of the customer is served by the facility to which he/she is *attracted* most (leading to Hotelling-type models) or *probabilistic*, when the customer splits his/her demand among all the existing facilities (leading to Huff-type models).

Furthermore, location problems can be defined as *pure location* problems, where the aim is to determine only the optimal sites for *nfc* new facilities ($nfc \geq 1$), or as *mixed* problems, in which, apart from the location, a decision has to be made about other variables. In mixed problems, besides the location of the *nfc* services, some "active" interactions have to be determined (see [127]). These interactions can be expressed by an *attraction (or utility) function* of a customer towards a given facility. For instance, in competitive location problems, it usually depends on the distance between the customer and the facility, and on other characteristics of the facility which determine its *quality*. The *market share* captured by the facilities depends on all those factors.

Finally, one may be interested in optimizing a single objective or several objectives simultaneously. This thesis deals both with single and multi-objective competitive continuous location problems. The former are usually *global optimization* problems, whereas the latter usually leads to *nonlinear multi-objective optimization* problems. In the following, a brief description of these fields is given.

1.1.1 Global optimization

The aim of *global optimization* is to find the best (global) solution of models, in the presence of (many) local and global optimal solutions. Formally, global optimization

seeks the global solution of a constrained optimization model. Nonlinear models are ubiquitous in many applications, e.g., in engineering design, biotechnology, data analysis, environmental management, financial planning, process control, risk management or scientific modeling, among others. Their solution often requires a global search approach.

The formulation of a global optimization problem, in its maximization form, is given by:

$$\begin{aligned} &\max f(y) \\ &\text{s.t. } y \in S \end{aligned} \tag{1.1}$$

where S is a nonempty closed set in $\mathbb{R}^n$ and f is a real valued function. Therefore, the objective is to find the maximum value f^* and all the points $y^* \in S$ such that:

$$f^* = f(y^*) \geq f(y) \quad \forall y \in S \tag{1.2}$$

The conversion of a minimization problem to a maximization one is straightforward

$$\min\{f(y)|y \in S\} = -\max\{-f(y)|y \in S\} \tag{1.3}$$

so, minimizing $f(y)$ is equivalent to maximizing $-f(y)$.

1.1.2 Nonlinear multi-objective optimization

A general nonlinear multi-objective optimization problem (MOP) can be formulated as follows:

$$\begin{aligned} \min \quad & \{f_1(y), \ldots, f_{l_{\max}}(y)\} \\ \text{s.t.} \quad & y \in S \subseteq \mathbb{R}^n \end{aligned} \tag{1.4}$$

where $f_1, \ldots, f_{l_{\max}} : \mathbb{R}^n \longrightarrow \mathbb{R}$ are $l_{\max}$ real-valued functions. Let us denote by $f(y) = (f_1(y), \ldots, f_{l_{\max}}(y))$ the vector of objective functions and by $Z = f(S)$ the image of the feasible region.

When dealing with multi-objective problems it is necessary to clarify what 'solving' a problem means. In the following, some widely known definitions are provided to explain the concept of solution of (1.4).

Definition 1. *A feasible vector $y^* \in S$ is said to be* efficient *iff there does not exist another feasible vector $y \in S$ such that $f_l(y) \leq f_l(y^*)$ for all $l = 1, \ldots, l_{\max}$, and $f_j(y) < f_j(y^*)$ for at least one index j ($j \in \{1, \ldots, l_{\max}\}$). The set PS of all the efficient points is called the* efficient set *or* Pareto-set. *If y_1 and y_2 are two feasible points and $f_l(y_1) \leq f_l(y_2)$ for all $l = 1, \ldots, l_{\max}$, with at least one of the inequalities being strict, then it is said that y_1* dominates *y_2.*

Efficiency is defined in the decision space. The corresponding definition in the criterion space is as follows:

Definition 2. *An objective vector $z^* = f(y^*) \in Z$ is said to be* non-dominated *iff y^* is efficient. The set PF of all non-dominated vectors is called the* non-dominated set *or* Pareto-front. *If y_1 and y_2 are two feasible points and y_1 dominates y_2, then it is said that $f(y_1)$ dominates $f(y_2)$.*

Ideally, solving (1.4) means obtaining the whole efficient set, that is, all the points which are efficient, and its corresponding Pareto-front. However, for a majority of MOPs, it is not easy to obtain an exact description of the efficient set or Pareto-front, since those sets typically include an infinite number of points (usually a continuum set). To the extent of our knowledge, only two exact general methods, namely, two interval branch-and-bound methods [62, 63] (see Subsection 1.2.1) have been proposed in literature which obtain an enclosure of those sets up to a pre-specified precision. Specifically, they offer a list of boxes (multi-dimensional intervals) whose union contains the complete efficient set (and their images, the corresponding Pareto-front) as a solution. However, they are time consuming. Furthermore, they have large memory requirements, so that only small instances can be solved with them.

The common approach in literature is to approximate the Pareto-front by a finite set of points. But then, a measure to quantify the effectiveness of such an approximation is needed. According to [161] there exist three main methods for the assessment and comparison of Pareto-set approximations:

- The *dominance ranking* method [67], which allows collections of Pareto-set approximations from two or more stochastic optimizers to be directly compared statistically. The dominance ranking approaches rely on the concept of Pareto dominance and some ranking procedure only. They yield general statements about the

relative performance of the algorithms which are independent of any preference information.

- The *attainment function* method [34, 66], which estimates the probability of attaining each goal in the objective space, and looks for significant differences between these probability density functions for different methods.
- The *quality indicator* method, the dominant method in literature, which maps each Pareto-front approximation to a number, and performs statistics on the resulting distribution(s) of numbers.

In this thesis, this last approach has been followed. Some definitions must be clarified.

Definition 3. *A feasible vector $y^* \in S$ is said to be* weakly efficient *iff there does not exist another feasible vector $y \in S$ such that $f_l(y) \leq f_l(y^*)$ for all $l = 1, \ldots, l_{\max}$. If y_1 and y_2 are two feasible points and $f_l(y_1) \leq f_l(y_2), l = 1, \ldots, l_{\max}$, then it is said that y_1* weakly dominates *y_2, and will be denoted by $y_1 \preceq y_2$.*

Now, the previous definition is extended to sets.

Definition 4. *It is said that set A* weakly dominates *set B, $A \preceq B$, provided that every point $y_2 \in B$ is weakly dominated by at least one point $y_1 \in A$.*

Another concept that will be used in the explanation of the algorithms presented in Chapter 4 of this thesis follows.

Definition 5. *It is said that two feasible points y_1 and y_2 are* indeterminate (or incomparable) *provided that neither $y_1 \preceq y_2$ nor $y_2 \preceq y_1$.*

The corresponding definitions apply in the criterion space. A general formal definition of a quality indicator follows.

Definition 6. *A unary* quality indicator *is a function $\mathcal{I} : SPF \rightarrow \mathbb{R}$ which assigns each Pareto-front approximation set $PF_{ap} \in SPF$ a real value $\mathcal{I}(PF_{ap})$.*

It is desired that whenever an approximation set A_{ap} of the Pareto-set is preferable to an approximation set B_{ap} with respect to weak Pareto dominance, the indicator value for $f(A_{ap})$ should be at least as good as the indicator value for $f(B_{ap})$. Such indicators are called *Pareto compliant*.

Definition 7. *A quality indicator is said to be* Pareto compliant *iff for any pair of approximation sets A_{ap}, B_{ap}, $A_{ap} \preceq B_{ap}$ implies that the indicator assigns a better (or equal) indicator value to $f(A_{ap})$.*

The most commonly used quality indicator in literature, and the one used in this thesis, is the *hypervolume* [158, 163]. This Pareto compliant indicator measures the hypervolume of the portion of the criterion space that is weakly dominated by the approximation set. The higher the hypervolume, the better the approximation. In order to measure this quantity, a reference point that is dominated by all points is needed. For a given problem, the same reference point has to be used for all the algorithms and all the runs. Assume that the quality of the outcomes generated by $q_{\max}$ stochastic algorithms must be compared. For each algorithm q, $q \in \{1, \ldots, q_{\max}\}$, rs_q runs are performed, generating the approximation sets $PS_1^q, \ldots, PS_{rs_q}^q$ (in the decision space). Lets denote by SPS the set of all the approximation sets of the Pareto set, $SPS = \{PS_1^1, \ldots, PS_{rs_1}^1, \ldots, PS_1^{q_{\max}}, \ldots, PS_{rs_{q_{\max}}}^{q_{\max}}\}$. In the computational studies of this thesis, the point whose l-th component is the maximum of all the l-th components of points in $f(SPS)$ is considered as reference point. It is an approximation of the Nadir point obtained when considering all the approximations of the Pareto-front together.

In Algorithm 1, a description of how to compute the hypervolume when two objective functions are considered, f_1 and f_2, is given. The first step is to compute the reference point $RP = (f_1^{(\max)}, f_2^{(\max)})$, where $f_l^{(\max)}$ denotes the maximum value of f_l with $l = 1, 2$ when considering all the solutions in SPS. Then, the hypervolume is calculated as the area covered by the points of the Pareto-front and the reference point RP. In Figure 1.1 a graphic representation of this metric is given for a bi-objective optimization problem. In particular, the figure on the left hand shows the calculation procedure of the hypervolume metric and the figure on the right illustrates how the hypervolume increases as the number of points in the Pareto-front does.

Hypervolume can be thought of as a *global* quality indicator, in the sense that it assesses the approximation as a whole. There are many other indicators in literature which are designed to assess a single aspect of the approximation (its proximity to the true Pareto-front, its spread, its evenness, etc.), although many of the them are not Pareto compliant. They are still useful, since they may refine the preference structure of Pareto compliant indicators. *Proximity* indicators, such as the (non Pareto compliant) *average distance* [33] and the (Pareto compliant) *unary additive epsilon indicator* [164],

Algorithm 1: Computation of Hypervolume indicator

1: Compute reference point $RP = (f_1^{(\max)}, f_2^{(\max)})$
2: Set $hyper = 0$
3: **for** $ic = 1$ to $(L_{\max} - 1)$ **do**
4: $\quad hyper = hyper + (f_2^{(\max)} - f_2^{(ic)}) \cdot (f_1^{(ic+1)} - f_1^{(ic)})$
5: $hyper = hyper + (f_2^{(\max)} - f_2^{(L_{\max})}) \cdot (f_1^{(\max)} - f_1^{(L_{\max})})$

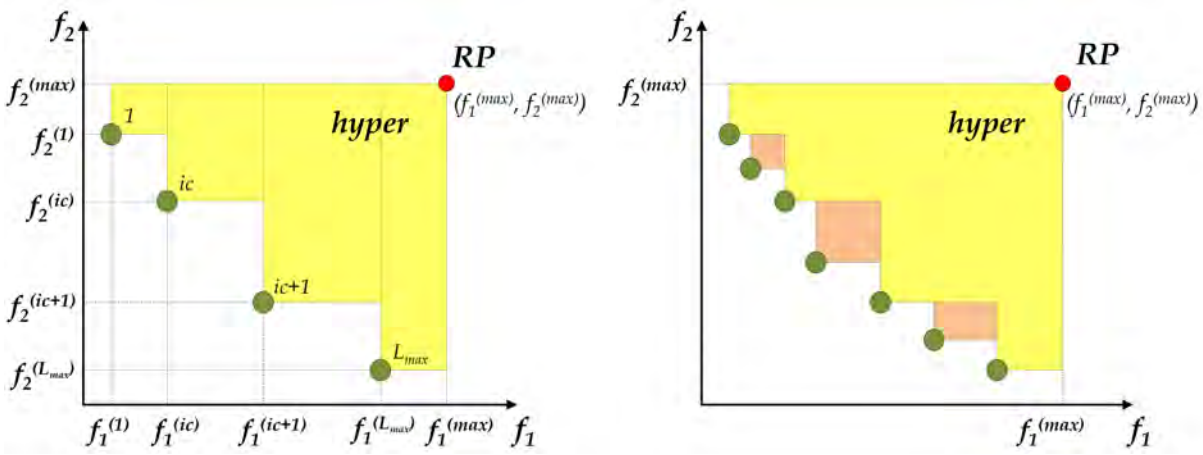

Figure 1.1: Hypervolume ($hyper$) calculation.

somehow measure the distance between the approximation set and true Pareto-front. Two (non Pareto compliant) *evenness/diversity* indicators are the *spread* [40, 121] and the *spacing* [151].

When computing quality indicators, normalized objective values are used to allow different objectives to equally contribute to comparative indicator values. The standard normalization is

$$f_l(y)' = \frac{f_l(y) - f_l^{(\min)}}{f_l^{(\max)} - f_l^{(\min)}},$$

where $f_l^{(\min)}$ (resp. $f_l^{(\max)}$) denotes the minimum (resp. maximum) value of f_l when considering all the solutions in SPS.

1.2 Search algorithms

In general, classical (i.e. local) optimization techniques have difficulties in dealing with global optimization problems. One of the main reasons is that they can easily be entrapped in local optima. Moreover, local optimization techniques cannot generate or even use the global information needed to find the global optimum for a function with multiple local optima. In order to find the global optimum, a global scope search effort is needed. The field of global optimization studies those problems of type (1.1) which have one or several global optima.

There are many taxonomies of global optimization strategies. An example can be found in [123], where optimization strategies are classified, according to the degree of rigor with which they approach the goal, into:

- *Incomplete methods*, where clever intuitive heuristics are used, but they cannot guarantee to escape from a local optimum, and hence, they cannot guarantee that the global optimum is found.

- *Asymptotically complete methods*. The probability of obtaining the global optimum is one if they run indefinitely. However, that a global optimum is reached if run a finite time cannot be ensured.

- *Complete methods*. They can reach the global optimum with certainty when exact computations are used and they run indefinitely. They can also find an approximate global optimum, within prescribed tolerances, after a finite time.

- *Rigorous methods*, where the global optimum is reached with certainty, within given tolerances, even with approximate computations.

Other classifications of global optimization methods can be consulted in [129, 152]. In spite of this, nowadays, it is really difficult to propose a classification of the search algorithms, since they blend different techniques in order to obtain the global optimum. Figure 1.2 depicts the classification followed in this work. Note that it focuses on *heuristics*, because of the stochastic nature of the algorithms developed in this thesis.

Initially, global optimization methods are divided into *exact* and *heuristic*, depending on whether they can guarantee the optimal solution has been found or not.

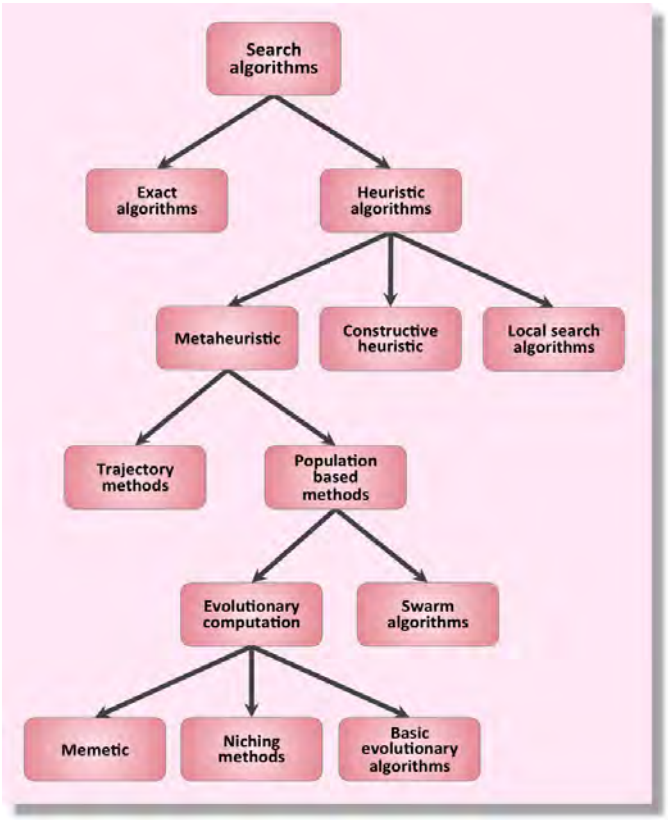

Figure 1.2: A taxonomy of search algorithms.

In *exact methods* (such as outer approximation, Lipschitzian optimization or Branch and Bound), random factors are hardly included. Some of them converge to the global optimum under some conditions, but when the algorithm is stopped after a finite number of iterations the accuracy of the solution may not be known with exactness. Exact methods with finite termination require detailed access to global information about the problem. More efficient complete methods generally combine branching techniques with one or several techniques from local optimization, convex analysis, interval analysis or constraint programming.

Moreover, exact algorithms are guaranteed to find, for every finite size instance of a problem, an optimal solution. Yet, for $\mathcal{NP}$-hard problems, no polynomial time algorithm exists. For practical purposes, exact methods are very expensive from a computational point of view, and heuristic methods have received more and more attention in the last few years. In these kind of methods, the guarantee of finding the

optimal solution is sacrificed for the sake of obtaining a good solution in a sufficiently short period of time.

Heuristics can provide useful and practical solutions for a wide range of problems and application domains. The power of heuristics comes from its ability to deal with complex problems when little or no information of the search space is given. They are particularly well suited to deal with a wide range of computationally intractable optimization and decision-making applications.

Heuristic methods can be considered algorithms that perform biased random searches of feasible solutions for a problem until a particular termination condition is met or after a predefined number of iterations is reached. *Heuristic* algorithms cannot guarantee that the set of found solutions includes the global optimum. Usually, these algorithms can find a solution whose objective value is close to the global optimum and they find it fast and easily. Sometimes these algorithms can be accurate, which means they actually find the best solution, but the algorithm is still called heuristic since this solution cannot be proven to be the best one.

Usually, heuristics imitate successful strategies found in nature. For example, evolutionary techniques copy the principles applied to species to develop superior qualities over generations; simulated annealing is based on how crystals emerge when materials are cooling and particles "find" a structure that minimizes the energy balance; some population based methods replicate the combined intelligence of crowds or the collective behavior of social animals, where the whole is more than the sum of its members' skills; etc.

The following subsections are devoted to briefly describe the exact and the heuristic branches of Figure 1.2.

1.2.1 Exact algorithms: iB&B

Among the exact algorithms, Branch-and-Bound (B&B) methods are probably the most successful ones. Many applications of these methods for solving many and diverse types of problems can be found in literature. See for instance [93] and references therein. They are deterministic algorithms based on search trees. The basic idea in B&B consists of a recursive decomposition of the original problem into smaller disjoint subproblems until the solution is found. A search tree, $T(V, \beta)$, describes these subproblems (V is the

set of vertices, whose root is the whole search space S) and the decomposition process (the set of arcs β). The algorithm avoids visiting subproblems which are known to not contain an optimal solution [25].

In [38] a general description of B&B methods is given and a common structure for all the applications, using general rules and operators [97, 114], is established. There are four main rules:

- *Division rule*: Establishes how the nodes of the search tree are decomposed.
- *Bounding rule*: Finds upper and/or lower bounds for the optimal solution of a node.
- *Selection rule*: Determines the next node which will be processed.
- *Elimination rule*: Recognizes and eliminates nodes where the solution of the original problem cannot lie.

Depending on the kind of problem, the following rule can be added:

- *Termination rule*: Determines if a node belongs to the final solution. This rule appears in problems where the termination criteria are given by some specified threshold between the best upper and lower bounds.

In B&B methods the feasible set is relaxed and subsequently partitioned in more and more refined parts (branching) over which feasibility conditions and upper and/or lower bounds of the objective function value can be determined (bounding). Parts of the feasible set with upper bounds not exceeding the best lower bound for the global optimum value found at a certain stage of the algorithm are deleted (pruning), since these parts of the domain cannot contain the optimum.

A good understanding of the problem to be solved is needed to select the previous basic rules correctly, since their good election can, on the one hand, decrease the execution time and, on the other hand, allow the resolution of harder problems. These basic rules are chosen depending on whether the search space is divided into general polygons or into other special sets, like triangles or squares (see [47, 86]). In fact, this division depends on how one can compute bounds for the given regions.

In chapters 2 and 4 of this thesis, B&B algorithms in which the bounds are automatically computed with the help of *Interval Analysis* have been used. It is only required that the search region be a closed set and that the functions to be optimized have a corresponding *inclusion function* (see below). Sometimes (twice) differentiability in some subregions is only required to accelerate the convergence. Some notation about interval analysis and related concepts are introduced in the following.

Real numbers will be denoted by lower case letters, $x, y, ...$, and intervals by capital letters, $X = [\underline{x}, \overline{x}]$, $Y = [\underline{y}, \overline{y}]$,..., where $\underline{x} = \min\{x \in X\}$ and $\overline{x} = \max\{x \in X\}$. The set of intervals will be denoted by $I(\mathbb{R}) := \{[a, b] \mid a \leq b; a, b \in \mathbb{R}\}$ and the set of n-dimensional interval vectors, also called *boxes*, by $I(\mathbb{R})^n$. For both real and interval vectors the following notation is used: $x = (x_1, x_2, ..., x_n)^T$, $x_i \in \mathbb{R}$ and $X = (X_1, X_2, ..., X_n)^T$, $X_i \in I(\mathbb{R})$, $(i = 1, 2, ..., n)$. The *width* of an interval X is defined by $w(X) = \overline{x} - \underline{x}$, if $X \in I(\mathbb{R})$ or by $w(X) = \max_{i=1,...,n} w(X_i)$ if $X \in I(\mathbb{R})^n$.

The main interval arithmetic tools applied to optimization methods are the *inclusion functions*. Let $D \subseteq \mathbb{R}^n$, let $I(\mathbb{R}^n)(D)$ denote the set of boxes in D, $I(\mathbb{R}^n)(D) = \{X : X \in I(\mathbb{R}^n), X \subseteq D\}$.

Definition 8. *Let $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$ be a real-valued function. A function $F : I(\mathbb{R}^n)(D) \to I(\mathbb{R})$ is said to be an inclusion function of f if:*

$$f(X) \subseteq F(X), \forall X \in I(\mathbb{R})(D)$$

The inclusion function generally overestimates the range of a function. The extent of the overestimation depends on the type of the inclusion function, on the considered function and on the width of the interval. If the computational costs are the same, the smaller the overestimation, the better the inclusion function is.

Interval branch-and-bound (iB&B) methods are based on *Interval Analysis*. Apart from using boxes to define the search region and its branches, they use *inclusion functions* to bound the objective function over a given box [25, 59, 153].

The prototype iB&B is shown in Algorithm 2. This algorithm selects the next box to be processed (*Selection rule*, line 3), which can be either totally or partially eliminated if it is known that the box does not contain any global maximizer (*Elimination rule*, line 5). This elimination process is based on some information obtained from inclusion functions (*Bounding rule*, line 4). If the box cannot be rejected, it is divided

Algorithm 2: A general interval B&B algorithm

1: Set the working list $List_w := \{S\}$ and the final list $List_f := \{\}$
2: **while** $List_w \neq \{\}$ **do**
3: Select a box X from $List_w$
4: Compute $F(X)$
5: **if** X cannot be eliminated **then**
6: Divide X into X^{ic}, $ic = 1, ..., ic_{\max}$
7: **for** $ic = 1$ to $ic_{\max}$ **do**
8: **if** X^{ic} satisfies the termination criterion **then**
9: Store X^{ic} in $List_f$
10: **else**
11: Store X^{ic} in $List_w$
12: Return $List_f$

(*Division rule*, line 6). The obtained subboxes are stored in the final list if they satisfy the termination criterion (*Termination rule*, line 9). Otherwise, they are stored in the working list to its posterior processing (line 11). When there are no boxes to be processed (i.e. $List_w$ is empty), the algorithm finishes (line 2) and returns the set of boxes in the solution list $List_f$ as solution.

A similar algorithm can be devised to cope with multi-objective problems. In that case, boxes are eliminated when it is known that they do not contain any efficient point, i.e., all their points are dominated. Also notice that for multi-objective problems the aim is to obtain the complete efficient set (and its corresponding Pareto-front), so the solution boxes offered by the algorithm contain the efficient set, and the images of those boxes contain the corresponding Pareto-front.

The iB&B method used in Chapter 2 of this thesis to solve the global optimization problem introduced there is described in [153]. The algorithm includes several new accelerating devices and modifies others presented in literature in order to solve difficult highly nonlinear problems more efficiently. For more details about the method, the interested reader is referred to [153]. On the other hand, for the bi-objective location problem introduced in Chapter 4, the iB&B method detailed in [63] is used.

1.2.2 Heuristic algorithms: Evolutionary computation

In computer science, there are two fundamental goals which algorithms have to reach: provable good or optimal solution quality and provable good run times. A heuristic is an algorithm that abandons one or both of these goals. For example, a heuristic usually finds pretty good solutions, but there is no proof the solutions could not become arbitrarily bad; or it usually runs reasonably quickly, but there is no argument that this will always be the case.

As mentioned previously, it just may happen that there is no interest in reaching the optimal solution because the size of the problem to be solved is beyond the computational limit of known optimal algorithms within the available computer time. Moreover, in other cases, the problem could be solved optimally but it can be considered that the effort (time, money, etc) required to find the optimal solution is not worth it. In such cases, a heuristic algorithm, which may hopefully find a feasible solution close to the optimum in terms of objective function value, can be used. In fact, it is often the case that a well-designed heuristic algorithm can give good quality (near-optimal) results.

The main advantages of heuristic algorithms are that such algorithms are (often) conceptually simpler and (almost always) much cheaper computationally than exact algorithms.

Among the heuristic methods, it is possible to distinguish between *constructive heuristics*, *local search methods* and *metaheuristics*.

- Constructive heuristics are algorithms that always take the best immediate or local solution while finding an answer. They generate solutions from scratch by adding opportunely defined solution components to an initially empty partial solution. This is done until a solution is complete or other stopping criteria are satisfied [3].

- A local search method employs the idea that a given solution s may be improved by making small changes. Those solutions obtained by modifying a solution s, are called neighbors of s, and the application of an operator that produces a neighbor s' is commonly called a *move*. A local search algorithm starts with an initial solution and moves from neighbor to neighbor as long as possible while increasing the objective function value. The main drawback of this strategy is its

difficulty to escape from local optima where the search cannot find any further neighbor solution that improves the objective function value.

- A metaheuristic can be defined as a high-level algorithmic framework or approach that can be specialized to solve optimization problems. It consists of an iterative process, which guides other heuristics in a search for feasible solutions. Metaheuristics are approximate and usually non-deterministic, and they are generally applied to problems for which there is no satisfactory problem-specific algorithm or heuristic; or when it is not practical to implement such a method. They are not problem-specific, but they may make use of domain-specific knowledge in the form of heuristics that are controlled by an upper level strategy.

Metaheuristics

The term *metaheuristic*, first introduced in [78], derives from the composition of two Greek words. *Heuristic* derives from the verb *heuriskein* which means "to find", while the prefix *meta* means "beyond, in an upper level". Some well-known algorithms are considered metaheuristic: random optimization, random-restart hill climbing, genetic algorithms, simulated annealing, tabu search, ant colony optimization, GRASP,... Innumerable variants and hybrids of these techniques have been proposed, and many more applications of metaheuristics to specific problems have been reported. This is an active field of research, with considerable literature, a large community of researchers and users, and a wide range of applications.

According to Figure 1.2, metaheuristics can be classified into *trajectory methods* and *population based methods*. In the first group, the search process is characterized by a trajectory in the search space. It is possible to distinguish between metaheuristics which follow one single search trajectory corresponding to a closed walk on the neighborhood graph, and metaheuristics which allow larger jumps in the neighborhood graph. Simulated annealing and tabu search are typical examples of trajectory methods. These methods usually allow moves to worse solutions to be able to escape from local optima.

Population-based metaheuristics (PBM) [15] are algorithms that work on a *set of solutions* (i.e. a population) at the same time rather than on a single solution. At first glance, it might be seen that this idea does not really provide anything new, since the

previous algorithms could run several times to increase the probability of arriving at the global optimum. But there is an additional component that can make population-based algorithms essentially different from other solving methods: the concept of *competition* among solutions in a population. That is, they simulate the evolutionary process of competition and selection so that the candidate solutions in the population fight for room in future generations. In this way, population-based algorithms provide a natural, intrinsic way for the exploration of the search space.

Among the population-based metaheuristics branch in Figure 1.2, it is possible to distinguish between *Evolutionary computation* and *Swarm algorithms*. The former will be studied next. The latter is out of the scope of this thesis, but the reader is referred to [3, 41, 42, 159] for an in-depth description of the algorithms and their applications.

Evolutionary Computation (EC) is a modern search technique which uses computational models of processes of evolution and selection [102, 111]. Concepts and mechanisms of Darwinian evolution and natural selection are encoded in evolutionary computation methods and used to solve problems in many fields of engineering and science [36]. According to Figure 1.2, it is possible to find *memetic algorithms* and *niching methods* in the evolutionary computation branch.

- Memetic algorithms (MAs) blend different search strategies in a combined algorithmic approach [116]. MAs are population-based metaheuristics, which means that they maintain a population of solutions for the problem at hand. Since it is possible to have either feasible or proto-solutions (structures that can be extended/modified to produce feasible solutions) or even unfeasible solutions (which can be "repaired" to restore feasibility), the term "solutions" is adopted here [119]. It is also assumed that both repairing or extension processes can be performed quite fast, so as to justify including them in the population. In the context of MAs, an *agent* represents a processing unit that can hold multiple solutions, and has problem-domain methods that help to improve them if required [115]. Each individual/agent represents a tentative solution/method to the problem under consideration. For further advice on the design of MAs, the reader is referred to [116, 117, 118, 119].

- Niching methods are based on the mechanics of natural ecosystems. In nature, animals compete to survive by hunting, feeding, grazing, breeding, etc., and dif-

ferent species evolve to fill each role. A niche can be viewed as a subspace in the environment that can support different types of life. A species is defined as a group of individuals with similar biological features capable of interbreeding among themselves but that are unable to breed with individuals outside their group. For each niche, the physical resources are finite and must be shared among the population of that niche. The size of each subpopulation will reflect the availability of resources in the niche. A niche is commonly referred to as an optimum of the domain, the fitness representing the resources of that niche. An important variety of niching methods have been reported in literature (see [10, 35, 37, 68, 94, 109, 112]).

Some of the most studied population-based methods are *Evolutionary Algorithms* (EAs). As was mentioned before, strong resemblance to biological processes as well as their initial applications for modeling complex adaptive systems influenced the terminology used by evolutionary computation researchers. It borrows a lot from genetics, evolutionary theory and cellular biology. Thus, a candidate solution to a problem is called an *individual*, while the entire set of current solutions is called a *population*. For some problem domains, a population may be broken into several *subpopulations* or *species*. The actual representation (encoding) of an individual is called its *genome* or *chromosome*. Each genome consists of a sequence of *genes*, i.e. attributes that describe an individual. A value of a gene is called an *allele*. When individual solutions are modified to produce new candidate solutions, they are said to be *breeding* and the new candidate solution is called an *offspring* or a *child*. During the evaluation of a candidate solution, it receives a grade called *fitness*, which indicates the quality of the solution in the context of a given problem; in optimization problems, the fitness is usually the objective function value at the candidate solution. When the current population is replaced by offspring, the new population is called a new *generation*. Finally, the entire process of searching for an optimal solution is called *evolution* [92].

Algorithm 3 describes the basic structure of an EA. Initially, a population of randomly generated individuals (or individuals obtained from other sources such as constructive heuristics) is created. The fitness is used to determine the relative merit of each individual. Once the initial population is obtained, an iterative process is carried out. At each iteration, a new offspring is generated using a recombination operator,

Algorithm 3: Evolutionary Algorithms (EA)

```
1: Generate an initial population
2: Evaluate the population
3: while termination conditions are not met do
4:    Recombine the population to obtain a new offspring
5:    Mutate the offspring to obtain a new population
6:    Evaluate the new population
7:    Select individuals from the new population to be considered in the next
      generation
8: Return best solution found
```

usually a two-parent or multi-parent crossover [12, 53]. Other techniques use population statistics for generating offspring [120, 150]. In order to introduce some *noise* in the search process for avoiding the convergence to local optima, a mutation operator is applied to the offspring individuals. In some applications, small random changes are used as a mutation mechanism. But, in other ones, it proved to be quite beneficial to use improvement methods to increase the fitness of individuals. EAs which apply a local search algorithm to each individual of a population can be thought of as *memetic algorithms*. While the use of a population ensures an exploration of the search space, the use of local search techniques helps to quickly identify "good" areas in the search space. Nevertheless, a premature convergence towards sub-optimal solutions can happen when applying local search. In order to avoid this drawback, there are, apart from the use of a random mutation operator, numerous ways of maintaining the population diversity. Some of these ways are *crowding* [101], *fitness sharing* [81] and *niching* [112]. Finally, the individuals compete, either only among themselves or also against their parents, to belong to the population at the next iteration. This is done by a selection scheme.

There exist similar techniques which differ in the implementation details and the nature of the particular problem, i.e. *Genetic programming* [106, 107], *Evolutionary programming* [54], *Evolution strategy* [13] and *Learning classifier system* [108]. Nevertheless, the most popular type of EA is the *Genetic algorithm* [79, 92].

Evolutionary algorithms can also be used to obtain a discrete approximation of the Pareto-front (and of the efficient set) of multi-objective optimization problems. The structure of the algorithm is the same as for single-objective optimization problems

(see Algorithm 3) but now the fitness associated to a solution is related (somehow) to its efficiency. The other difference is that since in multi-objective problems we look for a finite 'set' of points approximating the Pareto-front, the solution offered by the algorithms is a set of points, instead of a singleton.

1.3 High performance computing issues

Parallel computing operates on the principle that large problems can almost always be divided into smaller ones which can be solved concurrently ("in parallel"). Parallel computing is a research area which involves the study of both, hardware and software issues. The former is related to parallel architectures, while the latter has to do with parallel programming models. This section is devoted to providing a brief description of the state of the art in high performance computing; it covers aspects such as hardware and software models, performance evaluation and tools as well as a summary of the characteristics of the set of parallel and distributed computers which have been used to carry out the performance evaluation of the parallel algorithms proposed in this work.

1.3.1 Parallel architectures

Parallel computing is a set of systems, tools and techniques intended to achieve a substantial reduction of the computing time for long running applications. In many cases, parallel computing is the only way to tackle challenging problems which cannot be solved with the resources provided by a single computer. However, the best performance results are achieved when the modeler/programmer has a good knowledge of the underlying parallel hardware. In addition, the understanding of the characteristics of the different parallel architectures allows the programmers, on the one hand, to select the best architecture for the needs of the user and the kind of program, and on the other hand, to achieve better performance for the program, mainly due to the necessity of optimizations of the code to fully exploit the target system.

Several classification schemes for parallel computers have been defined, but the first and most commonly mentioned was proposed by Flynn in 1972 [65]. Flynn's taxonomy describes four possible models for combining data and instruction streams: *SISD* (Single Instruction, Single Data stream), *SIMD* (Single Instruction, Multiple Data stream),

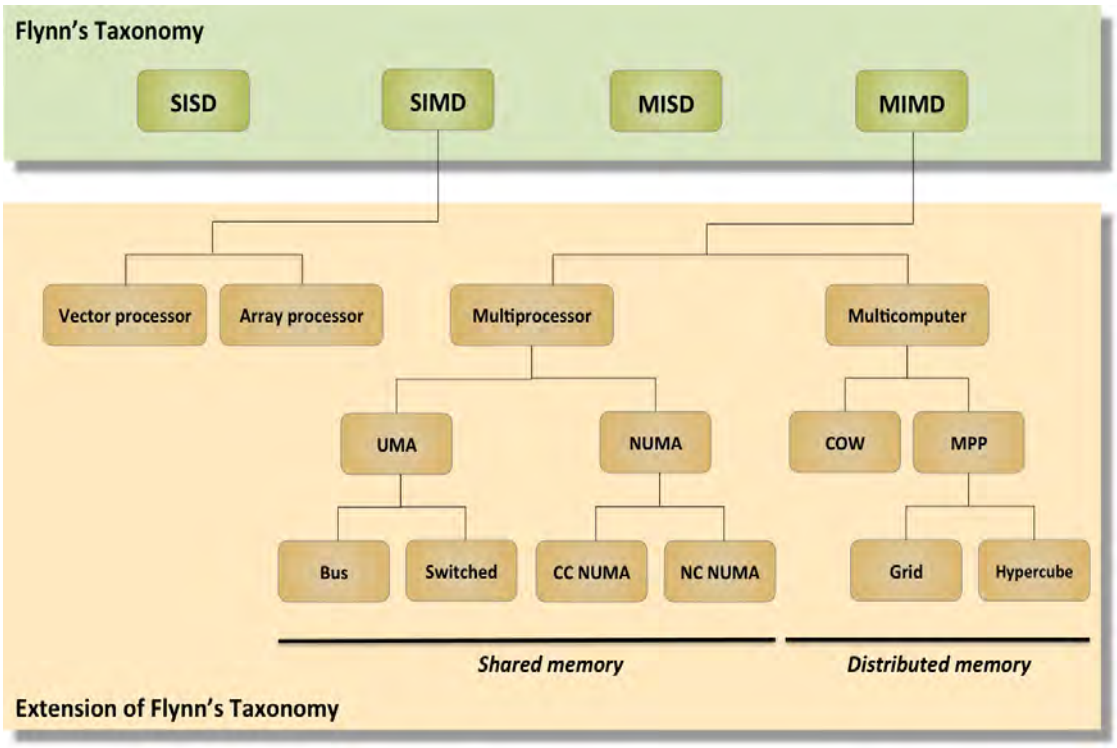

Figure 1.3: Extension of Flynn's Taxonomy.

MISD (Multiple Instruction, Single Data stream) and *MIMD* (Multiple Instruction, Multiple Data stream).

Although Flynn's taxonomy has become a standard model, the parallel computing evolution makes an extension necessary. In [16, 31, 52, 76, 90, 146] more exhaustive taxonomies are depicted. Figure 1.3 summarizes Flynn's taxonomy and the extension given by Alba in [3].

In *SISD* architecture, a single processor executes a single instruction stream, to operate on data stored in a single memory, i.e. it corresponds to systems that still have only one CPU (uniprocessor). However, *MISD* architectures refers to pipelined machines, where multiple instructions are executed on the same piece of data. This means that the parallelism comes from the own processor architecture. These two architectures refer, principally, to "old" personal computers, as nowadays, they include dual processor, etc. For this reason, they are out of the scope of this revision in parallelism.

SIMD architectures include the ones in which the same instruction is executed by all the processors, at each computing step or clock *cycle*, while MIMD comprise the

architectures where different data and programs can be loaded into different processors (processors work asynchronously and independently). This means that SIMD can be considered a particular case of MIMD architectures, where the execution of the same instructions is applied on different data.

Another type of parallel platform which is very interesting in this context is based on graphics processing units (GPUs). This type of architecture that had originally been designed for a specific purpose, is currently being exploited in general purpose computing (GPGPU) thanks to the availability of application programming interfaces (APIs) such as Compute Unified Device Architecture (CUDA) [145], AMD ATI Stream SDK [5] and OpenCL, considered the standard interface for programming accelerator units. From the point of view of the programmer, the GPU may be viewed as a set of SIMD multiprocessors with shared memory.

The next subection is dedicated to the most extended and used parallel architectures; i.e. MIMD architectures.

1.3.2 MIMD architectures

MIMD architectures can be divided into two main groups: *multiprocessors* and *multicomputers* (see Figure 1.3). Multiprocessor architecture refers to the model where all the processors have direct access to all the memory. Processors are connected to some interconnection network, through which they can access a common set of memory banks. Multicomputers refer to the architectures where each processor has its own local memory module. Processors access remote memory modules using an interconnection network and a message-passing method (see Subsection 1.3.3). The main multicomputer advantage is its "unlimited" scalability. In theory, a multicomputer can be composed of all the processors that are available. Nevertheless, note that now, the potential limitation may come from the interconnection network used to connect the computing nodes. Furthermore, multiprocessors are not scalable, but communications are transparent, in the sense that they take place via memory accesses.

Multiprocessors can be classified into two types: *centralized shared-memory* and *distributed shared-memory* architectures [90]. In the former, multiple processor-cache subsystems share the same physical memory, typically connected by one or more buses or a switch (see Figure 1.4). The key architectural property is the uniform access time

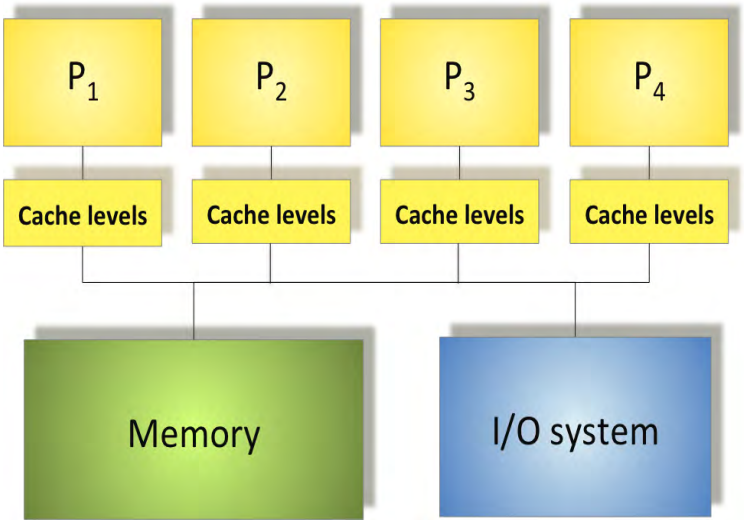

Figure 1.4: Uniform memory access architecture.

to all of the memory from all the processors. That is why these architectures are often called *uniform memory access* (UMA). The latter consists of individual nodes containing a processor, some memory, typically some Input/Output, and an interface to an interconnection network, which connects all the nodes (see Figure 1.5). In these architectures, the memory access time depends on the memory location relative to a processor. A processor can access its own local memory faster than non-local memory, that is, the local memory of another processor or the memory shared among processors. As memory accesses are not uniform, these architectures are called *non-uniform memory access* (NUMA). NUMA architectures may or may not maintain cache coherence across shared memory. If so, they are named CC-NUMA (*Coherent-Cache*), otherwise, they are called NC-NUMA (*Non Coherent-Cache*).

Multicomputers can be classified into (i) *COW* (cluster of workstations), where the system is composed of a *limited* number of computers interconnected by a communication network (see Figure 1.6), and (ii) *MPP* models (Massively Parallel Processors), which are composed of thousands of processors. MPP systems can be based on topologies such as *hypercube*, *fat tree* or *torus*, they can be tightly-coupled (i.e. it is a unique computer), or can be composed of machines belonging to multiple organizations and administrative domains, leading to the *grid systems*.

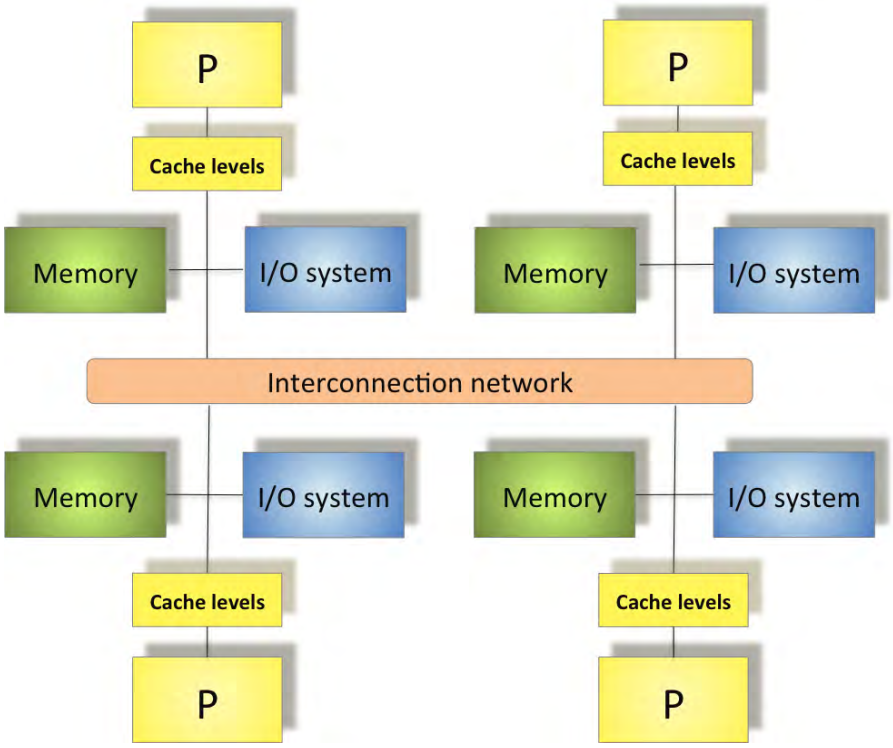

Figure 1.5: Non-uniform memory access architecture.

1.3.3 Parallel programming models and tools

A parallel program is intrinsically more complex than its serial counterpart. To write an efficient and scalable parallel program, one must understand the behavior and performance of the program. *Sequential programming* keeps a single process (i.e. a unique flow of control), while *parallel programming* uses *concurrent implementations*, where two or more processes work together to perform a single task and most of the times they need to communicate and synchronize among them. Communication and synchronization among different sub-tasks is typically one of the greatest barriers in obtaining good parallel program performance.

Concurrent programming includes both the programming of multiprocessors (*Shared memory programming*) and multicomputers or distributed systems (*Distributed memory programming*). In the next subsections these programming models are briefly described.

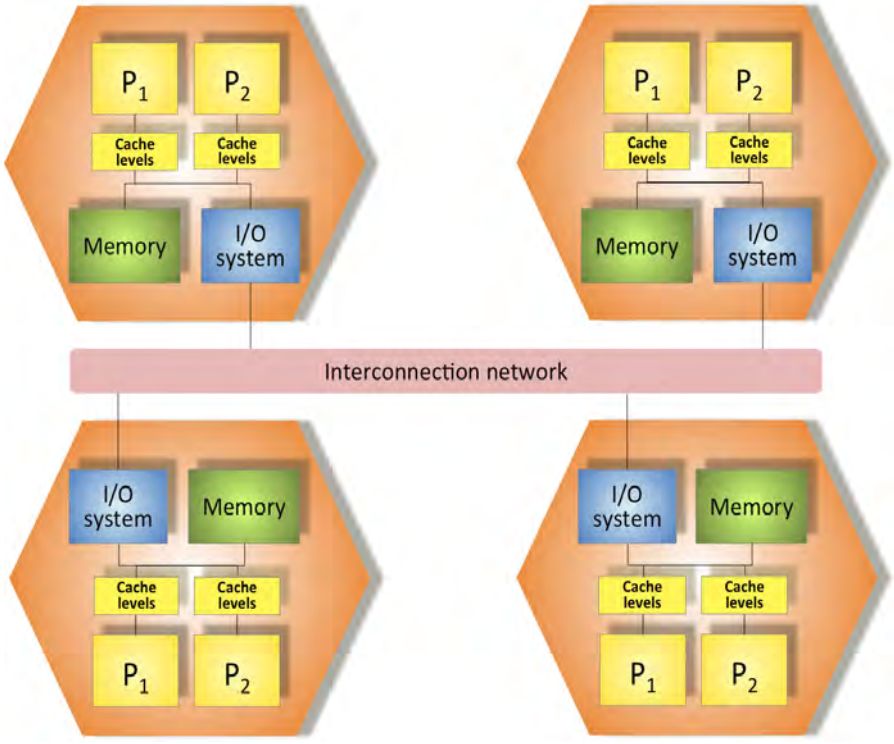

Figure 1.6: Multicomputer architecture, where each node is a multiprocessor.

Shared memory programming

In shared memory programming, the whole memory is directly accessible to all the processes with an intent to provide communication among themselves. Depending on context, programs may run on the same physical processor or on separate ones.

Multithreading (MT) is a programming paradigm for implementing application concurrency and, therefore, also a way to exploit the parallelism on shared memory multiprocessors [122]. A traditional "single threaded" can be defined as an independent flow of control associated one to one with a program counter, a stack to keep track of local variables, an address space and a set of resources. MT programming allows one program to execute multiple tasks concurrently, by dividing it into multiple threads, i.e. different streams of control that can execute its instructions independently and concurrently. This implies that the overlapping of input, output and computing operations is allowed. Moreover, when an MT program is executed on a multiprocessor machine, several threads can be run simultaneously (in parallel) on separate processors,

exploiting the parallelism of the hardware.

There exist several ways to deal with parallelism in a shared memory model, although the standardized library is *Pthreads* (or POSIX threads). Pthreads provides a unified set of C library routines with the main aim to make multithreaded implementations portable. This library includes functions for thread management (creation, scheduling and destruction) and synchronization (mutexes, synchronization variables, semaphores and read-write locks), and it is available mainly for several variants of the UNIX operating system.

Recently, modern programming languages, such as Java (`java.sun.com`), allow programmers work with multithreaded environments. There are two ways of creating threads in Java: implementing an interface and extending a class [43]. Extending a class is the way Java inherits methods and variables from a "single" parent class, while interfaces are used to design the requirements for "a set" of classes to implement. The interface sets everything up, and the class or classes that implement the interface do all the work. The different set of classes that implement the interface have to follow the same rules.

The *OpenMP* (Open Multi-Processing) (`www.openmp.org`) is an Application Programming Interface (API) that supports multi-platform shared memory multiprocessing programming in C/C++ and FORTRAN on many architectures (including Unix and Microsoft Windows platforms). It consists of a set of compiler directives, library routines, and environment variables that are used to express shared-memory parallelism. Jointly defined by a group of major computer hardware and software vendors, OpenMP is a portable, scalable model that gives programmers a simple and flexible interface for developing parallel applications. The programmers use the OpenMP directives to tell the compiler which parts of the program must be executed concurrently and to specify synchronization points.

Distributed programming

Distributed programming is based on the message-passing mechanisms. There exist several ways to interchange messages among processors, for example, the use of Internet computing system, object-based system and message-passing library. In the following some of the tools available to deal with distributed programming are described.

PVM (Parallel Virtual Machine) is a software package that permits a heterogeneous collection of computers hooked together by a network, to be used as a single large parallel computer [73]. Thus, large computational problems can be solved more cost effectively by using the aggregate power and memory of many computers. The software is very portable. The source, which is freely available through netlib [156], has been compiled on many computers, from laptops to CRAY multicomputer systems.

MPI (Message-Passing Interface) is a language-independent communications protocol used to program parallel computers [69]. Processes are written in a sequential language (C, C++, FORTRAN), and communications and synchronizations are made by calling functions from the MPI library. The MPI API is a consequence of people's experiences with earlier message-passing libraries, such as PVM. MPI can be considered a *de facto* standard, and there exist several implementations such as MPICH (`http://www.mpich.org`).

CORBA (Common Object Request Broker Architecture) is a middleware, which provides a distributed-object-based platform to develop distributed applications in which components can be written in different languages and can be executed in different machines [3].

Java RMI (Java Remote Method Invocation) enables the programmer to create distributed Java technology-based to Java technology-based applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts. RMI uses object serialization to marshal and unmarshal parameters and does not truncate types, supporting true object-oriented polymorphism. See [98] for more information.

The Globus Alliance is a community of organizations and individuals developing fundamental technologies that support grids and grid applications. Grid lets people share computing power, databases, instruments, and other on-line tools securely across corporate, institutional, and geographic boundaries without sacrificing local autonomy [77].

1.3.4 Parallel performance measures

One of the main goals in parallelism consists of increasing the performance of an application with respect to its execution on a uni-processor. The commonly used metric to

measure the performance of a parallel implementation on homogeneous processors is the *speedup*, which is defined as the ratio between the execution time on a uni-processor $T(1)$ and the execution time on P processors $T(P)$:

$$Spd(P) = \frac{T(1)}{T(P)} \tag{1.5}$$

Linear speedup (or ideal speedup) is obtained when $Spd(P) = P$. However, obtaining an ideal speedup is not always possible, since there are many factors which can increase the value of $T(P)$ and hence reduce the corresponding speedup. Among others, these factors can be: work load unbalance, sequential parts of the algorithm, communication overheads and synchronization among processors.

Nevertheless, sometimes a *superlinear speedup* can be obtained. This term refers to speedup larger than P when a parallel calculation is performed on P processors. According to Amdahl's Law [6] this is impossible, however, superlinear speedups have often appeared in literature [28, 141, 142, 148]. Many studies have been carried out in order to predict superlinear speedups or to determine the reason behind them (see [9] and the papers therein). Many of them coincide in that superlinear speedups can be obtained due to a suboptimal sequential algorithm, important changes which concern the performance of the parallel algorithms, or some unique feature of the architecture (memory, cache,...) that favors the parallel execution.

Another metric is *efficiency*, which estimates how well-utilized the processors are in solving the problem. So, the efficiency of a parallel version (run over P processors) with respect to the sequential one is computed as:

$$Eff(P) = \frac{Spd(P)}{P} = \frac{T(1)}{P \cdot T(P)} \tag{1.6}$$

Note that when linear speedup occurs efficiency is 1, which is called linear efficiency (or ideal efficiency). Besides, if superlinear speedup is obtained, the efficiency metric will be larger than 1. Figure 1.7 gives an example of the efficiency metric when all those different situations happen. In this thesis, the performance of all parallel implementations have been measured by the efficiency metric.

An important concept in parallelism is that of *scalability*. It can be understood as the ability of a system, network, or process, to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth. Using

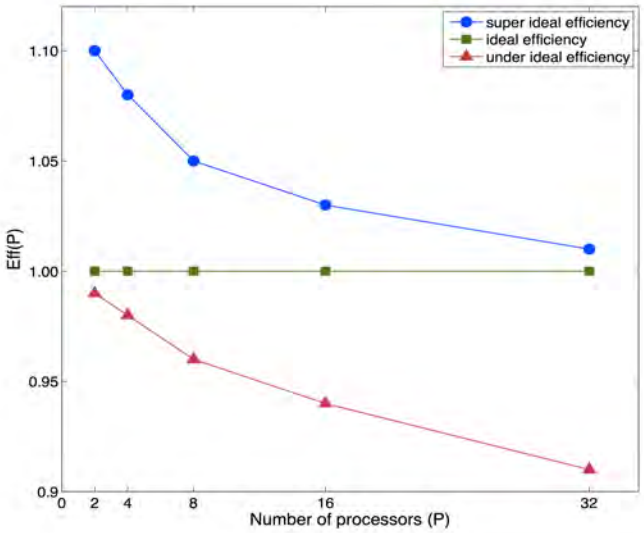

Figure 1.7: Efficiency example.

performance metrics (as speedup or efficiency) can determine whether an application is scalable or not. It is said that an application scales on an specific parallel architecture if it is suitably efficient and practical when applied to large situations (e.g. a large input data set, a large number of outputs or users, or a large number of participating nodes in the case of a distributed system). If the design or system fails when a quantity increases, it does not scale.

1.3.5 Computers and interfaces used in this thesis

All performance results shown later in chapters 2, 3 and 4 have been obtained from evaluations on a parallel architecture named *Ben Arabi*. *Ben Arabi* is the supercomputer of the Supercomputing Center of Murcia, Spain. It is divided into two machines:

- *Ben* is a cc-NUMA shared-memory architecture with 128 cores, organized into 16 cells with 8 cores each and 96GB of RAM. The processors are Intel Itanium-2 dual-core Montvale (1.6 GHz / 18MB of cache) and cells are connected by a crossbar network (see Figure 1.5).

- *Arabi* is a Blade Cluster with 816 cores, organized in 32 nodes with 16GB of memory each, and 70 nodes with 8GB each (102 nodes altogether). Each node has 8 cores, divided into 2 Intel Xeon Quad Core (E5450) to 3.0 GHz (COW, see Figure 1.6).

1.3.6 Parallel models in population-based methods

Population-based methods apply randomized operators over a population of candidate solutions to generate new points in the search domain. Nowadays, problems are *harder* and *larger*, and need larger population sizes to explore the search space deeply and then obtain good solutions. This translates directly into larger computational time and larger resource requirements (memory, processors...). In such a situation, a parallel machine together with a parallel model could be a good choice.

Literature contains many examples of successful parallel implementations. Some parallelization methods use a single population, while others divide the population into several relatively isolated subpopulations. Some methods can massively exploit parallel computer architectures, while others are better suited to multicomputers with fewer and more powerful processors. In this section three types of parallel methods are considered: (1) master-slave, (2) coarse-grain and (3) fine-grain.

Master-slave model

Master-slave is a communication model where one processor (the master) has unidirectional control over one or more processors (the slaves). This technique is called "global parallel model" too, since the management of the population is global (i.e. all the individuals in the population are considered when selection or crossover procedure is carried out). Usually, the master takes charge of performing it. In this model, the parallelism comes from the evaluation of the individuals in the population. This is because the fitness of an individual is independent of the rest of the population, and there is no need to communicate during this phase. The evaluation of individuals is parallelized by assigning a fraction of the population to each available processor. Communications occur only as each slave receives its subset of individuals for evaluation and when the slaves return the fitness values. If the algorithm stops and waits to receive the fitness values for all the population before proceeding to the next generation, then the global

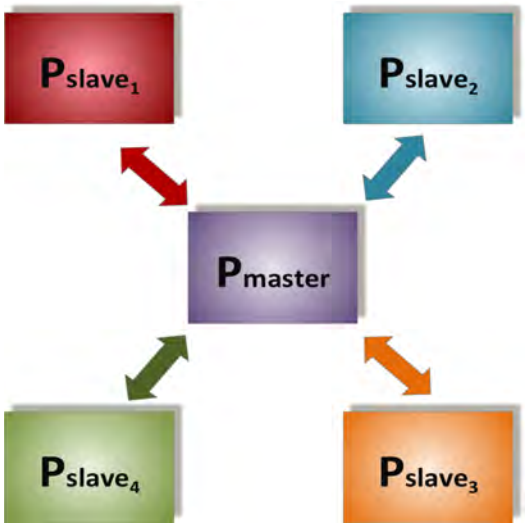

Figure 1.8: Master-slave model.

parallel algorithm is called *synchronous* and it has exactly the same properties as the sequential one. But, if the algorithm does not stop to wait for all processors, it does not work exactly like the sequential algorithm, then it is named *asynchronous* [24, 75].

Figure 1.8 depicts the communication procedure among processors. The central procesossor is the *master*, which interchanges some information with the *slaves*. Such interchanges are carried out using communications, that are represented by arrows.

The execution time of a master-slave model has three basic components: the time used in computations, the time used to communicate information among processors, and the waiting time due to the synchronization points. The first one is largely determined by the size of the population. However, the population size is also a major factor in the effectiveness of population-based methods and if the population is reduced, then the probability that the algorithm will find good solutions would decrease [80, 88]. The second one depends directly on the number of slaves, on the particular hardware used to execute the algorithm and on the size of the messages. Finally, the third one depends on the degree of parallelism of the algorithm, i.e. if there exist parts that have to be executed in a sequential way, and if the different parallel subtasks can evolve independently or, on the contrary, they need some information from the other ones to continue.

The master-slave method does not require a particular computer architecture, and

it can be implemented efficiently on shared and distributed memory computers. On a shared-memory multiprocessor, the population can be stored in shared memory and each processor could read a fraction of the population and write back the evaluation results. The number of individuals assigned to any processor can be constant, but in some cases (like in a multiuser environment where the utilization of processors is variable) it may be necessary to balance the computational load among the processors using a dynamic balancing algorithm (e.g., guided self-scheduling).

On a distributed-memory computer, the population is stored in one processor (the master), which is responsible for sending the individuals to the other processors (the slaves) for evaluation, collecting the results, and applying the genetic operators to produce the next generation. The difference with a shared-memory implementation is that the master has to send and receive messages explicitly.

It is important to emphasize that the master-slave method does not affect the behavior of the algorithm, i.e. the selection mechanism takes into account the entire population and it is possible to mate with any individual. However, there exist other methods which introduce fundamental changes in the way the algorithm works: selection is performed over a subset of individuals (subpopulation) and the population mating is restricted to the subpopulation. Such methods are applied to parallel strategies such as *coarse-grain* and *fine-grain* (see the following subsections.)

Coarse-grain model

In a coarse-grain model, each processor executes an algorithm independently of the remaining ones during most of the time [23]. The idea is that different processors work with smaller and different subpopulations in such a way that, when merging all the subpopulations, a population similar to that of the sequential version can be obtained. Nevertheless, some information can migrate from a processor to another one according to a migratory policy, which is controlled by the following parameters:

- *Interval of migration*: It establishes how often the migration of a certain amount of individuals will be conducted from a processor to another.

- *Rate of migration*: It indicates the number of individuals that have to communicate with other processors when the migration interval is fulfilled.

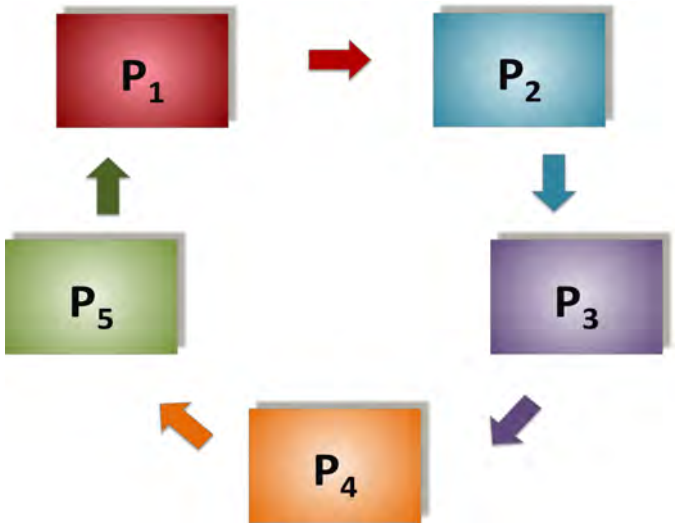

Figure 1.9: Coarse-grain model.

- *Selection criterion*: It determines the policy that will be applied for the selection of migratory individuals.

Figure 1.9 shows an example of migration policy, where processors are supposed to be connected in a ring topology. They communicate so that processor number i receives information from processor P_{i-1}, and sends it to processor number P_{i+1}.

Fine-grain model

This parallel model is suited for massively parallel computers and consists of one spatially-structured population [22]. Unlike a coarse-grain model, which consists of few and large subpopulations, a fine-grained model regards smaller subpopulations of which the larger ones are composed. The limit case is to have only one individual for every processor available. Additionally, selection and mating are restricted to a small neighborhood, but neighborhoods overlap permitting some interaction among all the individuals. See Figure 1.10 for a schematic of these types of algorithms, where squares represent processors and arrows refer to communications.

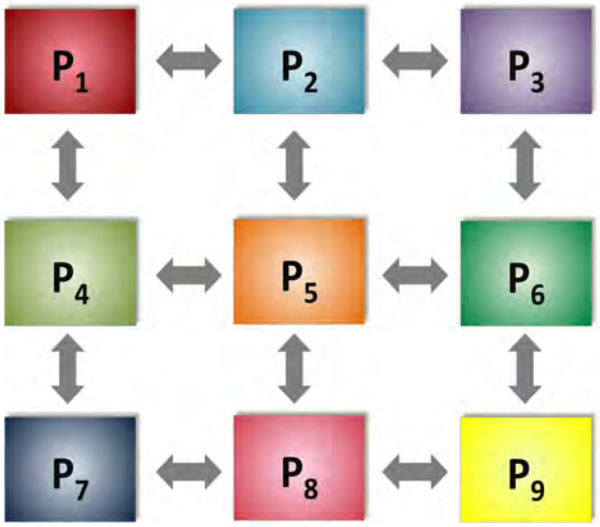

Figure 1.10: Fine-grain model.

Hybrid model

Nowadays, the tendency is to develop parallel algorithms which mix several techniques, in order to exploit all the available parallelism. Figure 1.11 shows an example of a hybrid model, which combines a coarse-grain model at the high level and a fine-grain strategy at the low level.

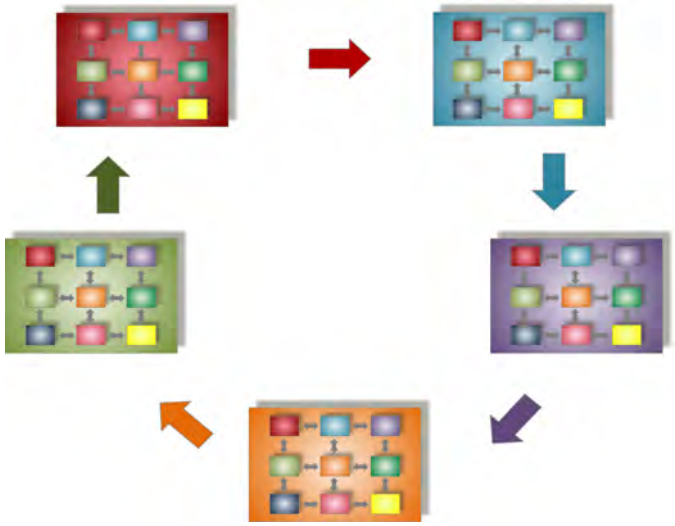

Figure 1.11: Coarse-grain model + fine-grain model.

Another way to hybridize is to use a coarse-grain method at both the high and the low levels. At the low level the migration rate is faster and the communications topology is much denser than at the high level (see Figure 1.12). A third method of hybridizing is to use a form of global parallelization (*master-slave*) on each of the subpopulations of a coarse-grain strategy (see Figure 1.13) [23].

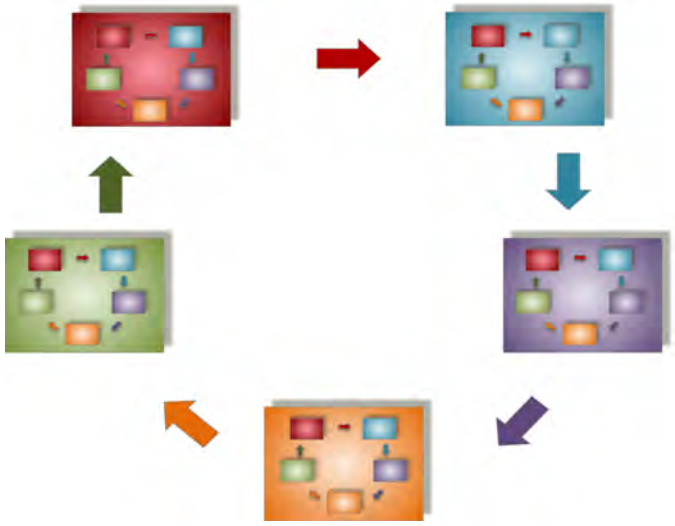

Figure 1.12: Coarse-grain model + coarse-grain model.

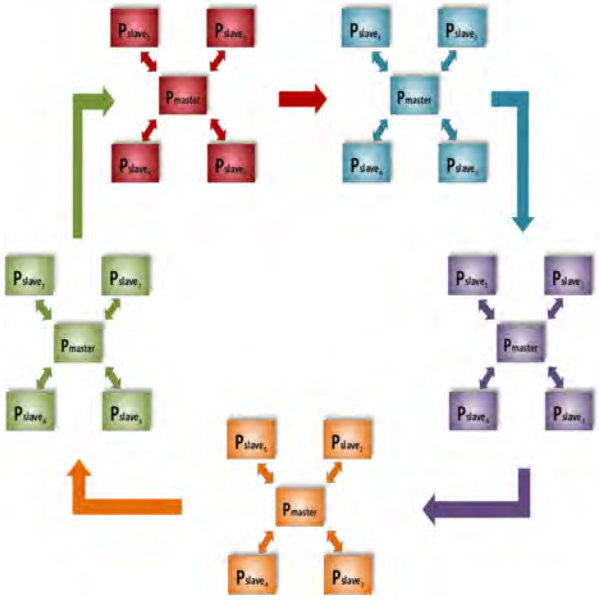

Figure 1.13: Coarse-grain model + master-slave model.

CHAPTER 2

A planar single facility location and design problem with endogenous demand

One of the major questions that a retail chain has to face when it considers entering or extending its presence in a market is 'where to locate' the new facility (or facilities) to be opened. If other facilities offering the same goods already exist in the area, the new facility will have to *compete* for the market. Many competitive location models are available in literature, see for instance the survey papers [56, 57, 128] and the references therein. In order to evaluate the market share resulting from the entry of the new facility, one needs to consider the way consumers choose facilities offering similar goods/services. Many quite different proposals exist in literature, as extensively explained in [48].

In most competitive location literature, it is assumed that the demand is exogenous (fixed) regardless the conditions of the market. Some remarkable exceptions are [11, 20, 83, 113]. Although this may be appropriate for essential goods, in other cases, this is mainly due to the difficulty of the problems to be solved: even with fixed demand, the corresponding location models may be hard-to-solve global optimization problems. However, sometimes demand is endogenous, that is, it varies depending on several factors. For instance, as already stated in [11], consumer expenditures on products or services offered by the facilities may increase for a variety of reasons related to location of the new facility: opening new outlets may increase the overall utility of the product; the marketing expenditures resulting from the new facilities may increase the overall 'marketing presence' of the product, leading to increased consumer demand; or some consumers who did not patronize any of the facilities, perhaps because none were close enough to their location, may now be induced to do so. On the other hand, the quality of the facilities may also affect consumer expenditures, since a better service usually leads to more sales.

Furthermore, to our knowledge, in none of the previous studies, has the effect of demand being influenced by facility layout been investigated. The first aim of this chapter is to study to what extent the optimal location *and quality* of new facilities to be located are affected by that assumption. In particular, it is considered a "spatial interaction model" recently proposed in literature [61, 135, 153] to analyze this effect. As will be shown, the corresponding problem with endogenous demand, introduced in Section 2.1, is much harder to solve. Section 2.2 will answer the question: *Exogenous or endogenous demand? Does it really matter when locating a facility?* In Section 2.3, two methods to cope with the problem at hand are investigated, an interval B&B method and an evolutionary algorithm called UEGO. A sensitivity analysis of the model is carried out in Section 2.4. The second aim of this chapter is to improve the efficiency of UEGO. In order to carry it out, the evolutionary algorithm has been slightly modified (see Section 2.5) and a parallelization of it is proposed in Section 2.6. Finally, some conclusions are pointed out in Section 2.7.

2.1 The model

A chain wants to locate a new single facility in a given area of the plane, where there already exist $j_{\max}$ facilities offering the same goods or product. The first k of those $j_{\max}$ facilities belong to the chain ($0 \leq k < j_{\max}$). The demand is supposed to vary with the location of the new facility (is *endogenous*) and is concentrated at $i_{\max}$ demand points, whose locations $locd_i$ are given, as well as the location $locf_j$ and quality of the existing facilities. The following notation will be used throughout this chapter:

Indices

- i index of demand points, $i = 1, \ldots, i_{\max}$.
- j index of existing facilities, $j = 1, \ldots, j_{\max}$.

Variables

- z location of the new facility, $z = (x, y)$.
- α quality of the new facility ($\alpha > 0$).
- nf variables of the problem, $nf = (z, \alpha)$.

Data

- $locd_i$ location of demand point i ($i = 1, \ldots, i_{\max}$).

$locf_j$	location of existing facility j ($j = 1, \ldots, j_{\max}$).
$d_{i,j}$	distance between demand point i and facility j.
$a_{i,j}$	quality of facility j as perceived by demand point i.
$g_i(\cdot)$	a non-negative non-decreasing function.
$u_{i,j}$	attraction that demand point i feels for facility j (or utility of j perceived by the people at i), $u_{i,j} = \dfrac{a_{i,j}}{g_i(d_{i,j})}$.
γ_i	weight for the quality of the new facility as perceived by demand point i.
$d_i^{\min}$	minimum distance from $locd_i$ at which the new facility can be located.
$\alpha^{\min}$	minimum level of quality for the new facility.
$\alpha^{\max}$	maximum level of quality for the new facility.
FR	region of the plane where the new facility can be located.

Miscellaneous

$d_i(z)$	distance between demand point i and the new facility.
$u_{i,nf}$	attraction that demand point i feels for the new facility, $u_{i,nf} = \dfrac{\gamma_i \alpha}{g_i(d_i(z))}$.
U_i	total utility perceived by demand point i provided by all the facilities.
$w_i(U_i)$	demand (or buying power or total expenditure) at demand point i.
$M(nf)$	market share captured by the chain.
$F(M(nf))$	expected sales obtained by the chain.
$G(nf)$	operating costs of the new facility.
$\Pi(nf)$	profit obtained by the chain.

It is assumed that $g_i(d_{i,j}) > 0 \ \forall i, j$. Following the framework of *spatial interaction models* introduced by Huff [96], we consider that the patronizing behavior of customers is probabilistic, that is, demand points split their buying power among all the facilities proportionally to the attraction they feel for them. The attraction that a demand point feels for a facility depends on both the location of the facility and its quality, as perceived by the demand point.

Furthermore, another assumption made is that the demand at $locd_i$ is affected by the perceived utility of the facilities, given by the vector $u_i = (u_{i,nf}, u_{i,1}, \ldots, u_{i,j_{\max}})$. Making the simplifying assumption that *the utility is additive*, then $U_i = u_{i,nf} + \sum_{j=1}^{j_{\max}} u_{i,j}$ represents the total utility perceived by a customer at $locd_i$ provided by all the facilities.

Hence, it is natural to assume that the actual demand at $locd_i$ *is a function of* U_i. Notice that this simplifying assumption still allows us to seek whether the optimal location and quality are affected by the type of demand, exogenous or endogenous, in the sense that if they are affected under this assumption then they will be affected in the more general case in which the utility is not additive.

If the maximum possible demand at $locd_i$ is denoted by $w_i^{\max}$, and the minimum possible demand at $locd_i$ by $w_i^{\min}$, then the actual demand w_i at $locd_i$ is a function of the utility vector u_i only through the total utility U_i, i.e., $w_i(U_i) = w_i^{\min} + incr_i \cdot e_i(U_i)$, where $incr_i = w_i^{\max} - w_i^{\min}$. Here, $e_i(U_i)$ is a non-negative and non-decreasing function of U_i that must not exceed 1 (notice that w_i cannot exceed $w_i^{\max}$). Function $e_i(U_i)$ can be interpreted as the share of the maximum possible increment that a customer decides to expend under a given location scenario.

There are different possible expressions for this. The following ones have been proposed in literature:

1. Linear expenditures: it is assumed that $w_i^{\min} = 0$, so that $incr_i = w_i^{\max}$. In this model w_i is represented by $w_i(U_i) = w_i^{\max} \cdot e_{i_1}(U_i)$, where $e_{i_1}(U_i) = c_i U_i$, with c_i a given constant such that $c_i \leq 1/U_i^{\max}$, where $U_i^{\max}$ is the maximum utility that can possibly be perceived by a customer at i, see [11].

2. Exponential expenditures: it is also assumed that $w_i^{\min} = 0$, so that $incr_i = w_i^{\max}$. In this model w_i is given by $w_i(U_i) = w_i^{\max} \cdot e_{i_2}(U_i)$, where $e_{i_2}(U_i) = 1 - e^{-\rho_i U_i}$, where $\rho_i > 0$ is a positive constant, see [11]. A similar model, with $e_{i_3}(U_i) = 1 - \rho_{i1}^{-\rho_{i2} U_i}$, where $\rho_{i1}, \rho_{i2} > 0$, was presented in [113], but notice that this is a special case of e_{i_2}, since $e_{i_3}(U_i) = 1 - e^{-\rho_{i2} U_i \ln(\rho_{i1})}$. This is a form of constant-elasticity demand function popular in economic literature.

We also suggest the following models:

3. Affine expenditures: $w_i(U_i) = w_i^{\min} + incr_i e_{i_1}(U_i)$. Notice that linear expenditures is a particular case of affine expenditures.

   In [11] there is another modification of linear expenditures, which is called bounded linear expenditures, where the value of $e_{i_1}(U_i)$ is fixed to a given constant above a given threshold.

4. $w_i(U_i) = w_i^{\min} + incr_i e_{i_4}(U_i)$, with $e_{i_4}(U_i) = \frac{1}{1+e^{(\rho_{i1}+\rho_{i2}\frac{1}{U_i})}}$, where $\rho_{i1} \in \mathbb{R}$ and $\rho_{i2} > 0$ are given constants. This last function has already been used in [51], and it allows to model different types of decreases in expenditures as the utility decreases. Notice that, unlike the previous e_i functions, e_{i_4} is not concave.

Based on these assumptions the market share captured by the chain is

$$M(nf) = \sum_{i=1}^{i_{\max}} w_i(U_i) \frac{u_{i,nf} + \sum_{j=1}^{k} u_{i,j}}{u_{i,nf} + \sum_{j=1}^{j_{\max}} u_{i,j}},$$

and the problem of profit maximization is described by

$$\begin{cases} \max & \Pi(nf) = F(M(nf)) - G(nf) \\ \text{s.t.} & d_i(z) \geq d_i^{\min} \ \forall i \\ & \alpha \in [\alpha^{\min}, \alpha^{\max}] \\ & z \in FR \subseteq \mathbb{R}^2 \end{cases} \tag{2.1}$$

where $F(\cdot)$ is a strictly increasing differentiable function which transforms the market share into expected sales, $G(nf)$ is a differentiable function which gives the operating cost of a facility located at z with quality α, and $\Pi(nf)$ is the profit obtained by the chain. The parameter $d_i^{\min} > 0$ is a given threshold, which guarantees that the new facility is not located on top of demand point i (although due to demand aggregation (see [70]) $locd_i$ is a point which usually represents a set of customers who occupy a given area). The parameters $\alpha^{\min}$ and $\alpha^{\max}$ are the minimum and maximum values, respectively, that the quality of a facility may take in practice. By FR we refer to the region of the plane where the new facility can be located.

In this thesis, function F is assumed to be linear, $F(M(nf)) = s \cdot M(nf)$, where s is the income per unit of goods sold. Function G should increase as z approaches one of the demand points, since it is rather likely that the operational cost of the facility will be higher around those locations (due to the value of land and premises, which will make the cost of buying or renting the location higher). On the other hand, G should be a convex function in the variable α, since the more quality we expect from the facility the higher the costs will be, at an increasing rate. It is assumed that G is a separable function, in the form $G(nf) = G^a(z) + G^b(\alpha)$, where $G^a(z) = \sum_{i=1}^{i_{max}} \Phi_i(d_i(z))$, with $\Phi_i(d_i(z)) = Aver_{A_i}(w_i(U_i))/((d_i(z))^{\phi_{i,0}} + \phi_{i,1})$, $\phi_{i,0}, \phi_{i,1} > 0$, and

$G^b(\alpha) = e^{(\frac{\alpha}{\alpha_0}+\alpha_1)} - e^{\alpha_1}$, with $\alpha_0 > 0$ and α_1 given values. Notice that in the cost function $G(nf)$, $Aver_{A_i}(w_i(U_i))$ stands for the average value of $w_i(U_i)$ over the feasible set (the mathematical formulation is given in Section 2.2, see Equation (2.2)) and can be thought of as an estimation of the demand at $locd_i$ by a fixed number. When using $Aver_{A_i}(w_i(U_i))$ instead of $w_i(U_i)$ in function G, we assume, on the one hand, that the cost of obtaining a given level of quality, as given by G^b, does not depend on the level of demand in the market. This can be realistic in many cases, especially when $incr_i$ is not too high. On the other hand, it also implies that the location cost does not depend on the level of demand either. This is especially true if the cost of buying or renting the place for the location is paid in advance, before opening the new facility. In this way, the scenario which determines the cost of the location is not affected by the 'variation' in the demand produced by the location of the new facility, but just by the expected average demand. Other possible expressions for F and G can be found in [61, 153].

Notice also that in this planar Huff-like competitive location and design model with endogenous demand, the location and the quality of the new facility are the variables of the problem, because although in most literature only the question of location is researched, these two features cannot be separated (see [154]).

2.2 Exogenous or endogenous demand? A key point to be taken into account

To be able to compare the solutions of the problems with exogenous (fixed) demand with the corresponding problems with endogenous (variable) demand we have to assign, given $w_i(U_i)$, a value to the estimated fixed demand at $locd_i$, which will be denoted by $\widehat{w}_i$. A suitable choice could be to set $\widehat{w}_i$ equal to the mean value of $w_i(U_i)$.

In the case studies presented in this chapter, and to simplify the computations, it is assumed that the region of the plane FR where the new facility can be located is a rectangle. Let us denote $r_i = \sum_{j=1}^{j_{\max}} u_{i,j}$. Then the range for U_i is the interval

$$A_i = \left[\frac{\gamma_i \alpha^{\min}}{g_i(d_i(V_i))} + r_i, \frac{\gamma_i \alpha^{\max}}{g_i(d_i^{\min})} + r_i \right]$$

where $d_i(V_i)$ is the distance between demand point $locd_i$ and the furthest feasible point

(since our searching region is a rectangle, the furthest point is the furthest of the vertices of FR, V_i). The mean value of $w_i(U_i)$ over A_i, denoted by $Aver_{A_i}(w_i(U_i))$, is given by

$$Aver_{A_i}(w_i(U_i)) = w_i^{\min} + incr_i \cdot Aver_{A_i}(e_i(U_i)),$$

where the mean value of the function $e_i(U_i)$, as a function of the variable U_i alone, over the interval A_i, is given, according to the well known first mean value theorem for integration, by

$$Aver_{A_i}(e_i(U_i)) = \frac{1}{\frac{\gamma_i \alpha^{\max}}{g_i(d_i^{\min})} - \frac{\gamma_i \alpha^{\min}}{g_i(d_i(V_i))}} \int_{\frac{\gamma_i \alpha^{\min}}{g_i(d_i(V_i))}+r_i}^{\frac{\gamma_i \alpha^{\max}}{g_i(d_i^{\min})}+r_i} e_i(U_i) dU_i. \tag{2.2}$$

The previous reasoning also gives us a hint about how to compute the value of the parameters $incr_i$,

$$incr_i = \frac{Aver_{A_i}(w_i(U_i)) - w_i^{\min}}{Aver_{A_i}(e_i(U_i))}. \tag{2.3}$$

Thus, to have a fair comparison between both location models and to solve a given problem under both scenarios, fixed and variable demand, given the problem with fixed demand (hence, given $\widehat{w}_i$), we could do the following:

1. Determine the function range $A_i = [\frac{\gamma_i \alpha^{\min}}{g_i(d_i(V_i))} + r_i, \frac{\gamma_i \alpha^{\max}}{g_i(d_i^{\min})} + r_i]$ of U_i.
2. Compute $Aver_{A_i}(e_i(U_i))$ using Equation (2.2).
3. Determine $incr_i$ according to Equation (2.3), and setting $Aver_{A_i}(w_i(U_i)) = \widehat{w}_i$ (notice that $w_i^{\min}$ must be less than $\widehat{w}_i$).

In order to compare both models (with endogenous and exogenous demand), firstly we have applied the previous process to two different instances, assuming in both cases (as will be done in the rest of the thesis) linear expenditures. The first one, with $i_{\max} = 71$ demand points, $j_{\max} = 5$ existing facilities, all belonging to the competitors ($k = 0$) corresponds to a quasi-real case of location of supermarkets in the south-east of Spain, see [154]. The second one, with setting ($i_{\max} = 50, j_{\max} = 2, k = 2$), was randomly generated as will be described in Subsection 2.3.4.

Figures 2.1 and 2.2 give the graphs of the objective function on the location domain when quality α is fixed. The white holes in the graphs correspond to the forbidden

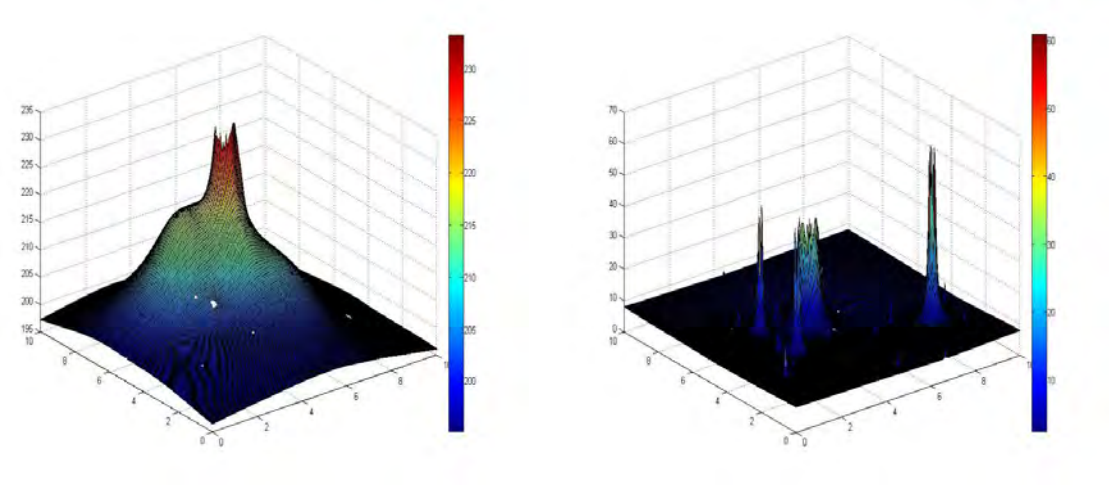

Figure 2.1: Objective function of the instance with setting $(i_{\max} = 71, j_{\max} = 5, k = 0)$ and $FR = ([0, 10], [0, 10])$, projected in the 2-dimensional location space (with $\alpha = 5.0$). (a) On the left with fixed demand (b) On the right with variable demand.

regions around the demand points, as given by the constraints $d_i(z) \geq d_i^{\min}$ $(i = 1, \ldots, i_{\max})$, which guarantee that the new facility is not located on top of the area where the customers represented by $locd_i$ lie. The two problems using the same data give a completely different shape of the objective function. In the fixed demand cases, the objective function has several local optima, but it tends to be smoother. In the variable demand cases, the objective function has many more local optima, and the landscape is much steeper. This effect can also be seen in Figure 2.3, where the contours of the objective function change from blue colours (low objective function value) to red colours (high objective function value) in fewer units of distance in the variable demand case.

Finally, to corroborate this fact, we have solved 41 instances, one for each of the settings $(i_{\max}, j_{\max}, k)$ in Table 2.1, assuming that $d_i^{\min} = 0.001\ \forall i$, using a local search, namely, a Weiszfeld-like algorithm (see [61] for fixed demand and Subsection 2.3.2 for variable demand). For each instance the local search was run 5000 times starting from different random points, and we have counted the number of different solutions (local optima) found in those runs. The number of different optima has been computed in two different ways. In the first one, we say that two solutions (x^1, y^1, α^1) and (x^2, y^2, α^2) are different provided that $\|(x^1, y^1, \alpha^1) - (x^2, y^2, \alpha^2)\|_2 > 2 \cdot d_i^{\min}$. Thus, we count the number of 'hills' of the objective function, since two local optima on the top of the same hill

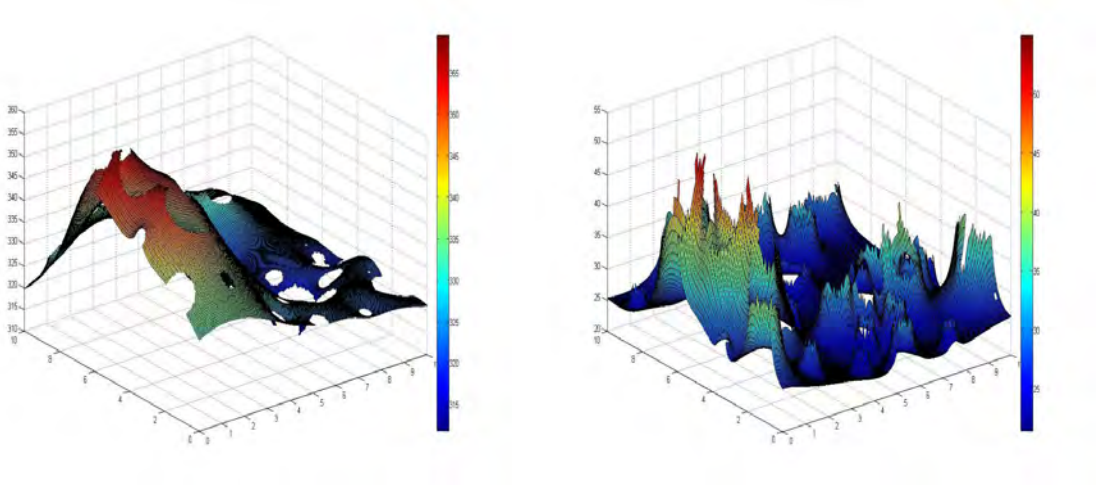

Figure 2.2: Objective function of the instance with setting $(i_{\max} = 50, j_{\max} = 5, k = 2)$ and $FR = ([0, 10], [0, 10])$, projected in the 2-dimensional location space (with $\alpha = 2.11$). (a) On the left with fixed demand. (b) On the right with variable demand.

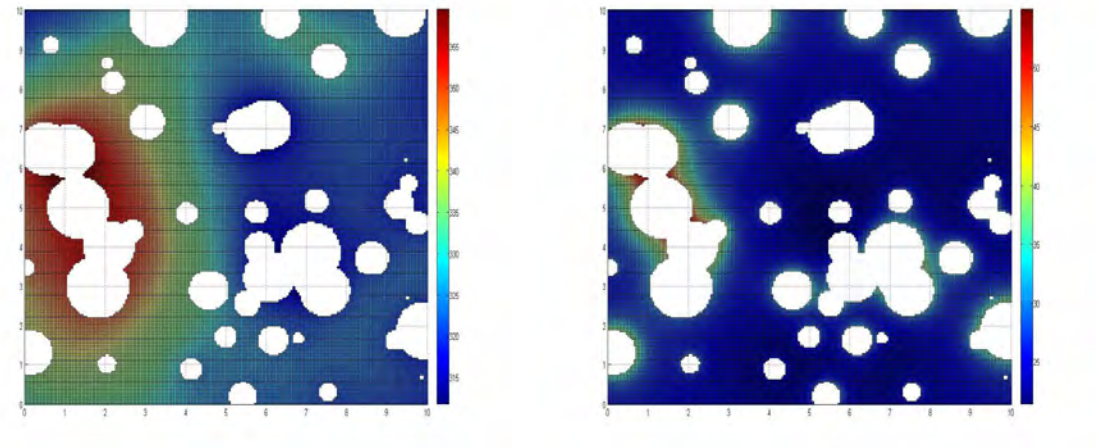

Figure 2.3: Contours of the objective function of the instance with setting $(i_{\max} = 50, j_{\max} = 5, k = 2)$ and $FR = ([0, 10], [0, 10])$, projected in the 2-dimensional location space (with $\alpha = 2.11$). (a) On the left with fixed demand. (b) On the right with variable demand.

over a demand point are considered equal. In the second one, two solutions (x^1, y^1, α^1) and (x^2, y^2, α^2) are considered different provided that $\|(x^1, y^1) - (x^2, y^2)\|_2 > 2 \cdot d_i^{\min}$. Therefore, two local optima on the boundary of the forbidden region of a demand point are considered equal. Notice that any demand point usually generates a hill, since the demand captured by the new facility usually increases as the new facility gets closer to

$i_{\max}$	50			100			200			500			1000		
$j_{\max}$	2	5	10	2	5	10	2	10	15	2	15	25	5	25	50
k	0,1	0,1,2	0,2,4	0,1	0,1,2	0,2,4	0,1	0,2,4	0,5,10	0,1	0,5,10	0,7,15	0,1,2	0,7,15	0,15,30
FR	$([0,10],[0,10])$									$([0,25],[0,25])$			$([0,50],[0,50])$		

Table 2.1: Settings of the test problems.

the demand point. This effect is more remarkable in the variable demand model, since in addition to this, the level of demand at a demand point also increases as the new facility gets closer to it.

The results are shown in tables 2.2 and 2.3, respectively, grouped according to the number of demand points. The average number of different solutions found for both the variable and fixed demand models are given first. Then, we give the minimum, average and maximum values of the difference between the number of different solutions. As we can see, the number of local optima is between 3.5 to 10.4 times greater in the variable demand problems as compared to the fixed demand problems when considering the three variables, and between 2.9 to 7.7 when considering only the location variables, thus confirming that the variable demand problem is more challenging from an optimization point of view.

Continuing with the two examples mentioned above, if we denote the optimal solution of the fixed demand instance by $(\check{x}, \check{y}, \check{\alpha})$, and the optimal solution of the corresponding variable demand instance by (x^*, y^*, α^*), we have that $(\check{x}, \check{y}, \check{\alpha}) = (5.13, 5.92, 0.50)$ and $(x^*, y^*, \alpha^*) = (8.45, 2.99, 5.00)$, with $\Pi(\check{x}, \check{y}, \check{\alpha}) = 282.46$ and $\Pi(x^*, y^*, \alpha^*) = 67.09$ in the first example. For the second example, the corresponding figures are $(\check{x}, \check{y}, \check{\alpha}) = (0.76, 5.86, 1.34)$, $(x^*, y^*, \alpha^*) = (0.73, 5.88, 2.11)$, $\Pi(\check{x}, \check{y}, \check{\alpha}) = 363.10$ and

	Endogenous	Exogenous	Difference		
$i_{\max}$	demand	demand	Min	Av	Max
50	3597	346	2635	3227	3769
100	3398	351	2368	3030	3442
200	3334	451	1860	2855	3612
500	1723	394	827	1317	1588
1000	1491	424	550	1067	1466

Table 2.2: Number of local optima considering the three variables (x, y, α).

	Endogenous	Exogenous	Difference		
$i_{\max}$	demand	demand	Min	Av	Max
50	2630	341	1688	2255	2846
100	2383	348	1348	2006	2496
200	2268	443	841	1797	2489
500	1249	389	353	844	1145
1000	1225	420	270	805	1168

Table 2.3: Number of local optima considering location variables (x, y).

$\Pi(x^*, y^*, \alpha^*) = 56.08$. These values have been obtained using the exact interval branch-and-bound method that will be outlined in Section 2.3 (see [153]). We can see that not only are the optimal values rather different, but also, and most importantly, the locations and the qualities of the new facilities to be located are rather different. And this is due to the variability of the demand. For instance, in the second example, for which $(\check{x}, \check{y})$ is similar to (x^*, y^*), the market share captured by the facility in the fixed demand case is 141.54, whereas this value goes down to 39.82 for the variable demand case.

It may be argued that the procedure followed here is not the best one. For instance, after setting $Aver_{A_i}(w_i(U_i)) = \widehat{w}_i$ and $c_i = \frac{1}{U_i^{\max}}$, when Equation (2.3) is used, $incr_i$ is computed (or equivalently, the upper bound $w_i^{\max}$ is computed) so that everything fits. Instead, for each demand point i, the average demand $\widehat{w}_i$, the minimum possible demand at it, $w_i^{\min}$, and the maximum possible demand at it, $w_i^{\max}$, could be estimated. And then, suitable values for the parameters of the $e_i(U_i)$ functions could be found so that $Aver(w_i(U_i)) = \widehat{w}_i$ holds. But in any case, the same conclusion can be derived. In fact, if a real problem has endogenous demand, whatever the value is assigned to $\widehat{w}_i$ when solving the problem with a fixed demand model, the solution will be wrong. Notice that the value assigned to $\widehat{w}_i$, will be assigned assuming a given location scenario (for instance, assuming that the new facility will be located at a medium distance from $locd_i$). But this will be done *for all* the demand points. So, regardless the value of nf, it cannot satisfy *all* those assumptions simultaneously. This means that when solving the problem with a fixed demand model, the overall demand will be overestimated, and hence, the final solution will be wrong.

Hence, we can conclude that the type of demand is a key point to be taken into

account when modeling a location problem, since both the optimal solution and the optimal value of the corresponding location problems may be quite different. Furthermore, the cost of assuming that the demand is fixed when it actually varies may be high (see Subsection 2.4.4).

2.3 Solving the location model

As we have seen in Section 2.2, the location model with endogenous demand may have many more optima than the corresponding exogenous demand model. So we need efficient algorithms for solving these types of problems. In [61], an exact interval branch-and-bound method was sought to solve the corresponding exogenous demand model. The same method can handle the endogenous demand model thanks to the use of the interval tools employed to compute the bounds. However, only instances with up to $i_{\max} = 200$ demand points can be solved with this exact method (see Subsection 2.3.4).

For bigger problems heuristic, procedures are needed. In [135], an evolutionary algorithm called UEGO was studied for solving the corresponding fixed demand mode. It proved to be able to generate solutions for instances with up to $i_{\max} = 10000$ demand points in around 8 hours of CPU time. Next, a modification of that method for the problem with endogenous demand is presented.

2.3.1 UEGO

Universal Evolutionary Global Optimizer (UEGO) is a multimodal algorithm which is able both to solve multimodal optimization problems where the objective function has multiple local optima and to discover the structure of these optima as well as the global optimum (see [99, 124, 126]). In a multimodal domain, each peak can be thought of as a niche. The analogy with nature is that within the environment there are different subspaces, niches, that can support different types of life (species or organisms). Niching, clustering or speciation methods are techniques that promote the formation and maintenance of subpopulations in the Evolutionary Algorithms (EA) (see Subsection 1.2.2).

The concept of niche is renamed *species* in UEGO. A species can be thought of as a window on the whole search space. This window is defined by its center and a radius.

The center is a solution, and the radius indicates its attraction area, which covers a region of the search space and hence, multiple solutions (see Figure 2.4). The radius of the species is neither constant along the execution of UEGO nor the same for each species. This radius is a monotonous function that decreases as the index level (or cycles or generations) increases. The parameter $t_{\max}$ indicates the maximum number of levels in the algorithm. The radius of a species created at level t (with $t \in [1, t_{\max}]$), is given by a decreasing exponential function (see [100]) which depends on the initial domain landscape (the radius at the first level, R_1) and the radius of the smallest species $R_{t_{\max}}$.

During the optimization process, a list of species is kept by UEGO. This concept, *species-list*, would be equivalent to the term population in an evolutionary algorithm. UEGO is in fact a method for managing this *species-list* (i.e. creating, deleting and optimizing species). The maximum length of the species list is given by the input parameter $L_{\max}$ (maximum population size).

UEGO has an upper bound $FE_{\max}$ on the number of function evaluations. However, it is important to mention that UEGO may terminate simply because it has executed all of its levels. The final number of function evaluations depends on the complexity of the objective (fitness) function. This is qualitatively different from other evolutionary algorithms, which typically run up to a limit on the number of function evaluations.

In UEGO every species is intended to occupy a local maximizer of the fitness function, without knowing the total number of local maximizers in the fitness landscape. This means that when the algorithm starts it does not know how many species there will be at the end. For this purpose, UEGO uses a non-overlapping set of species which

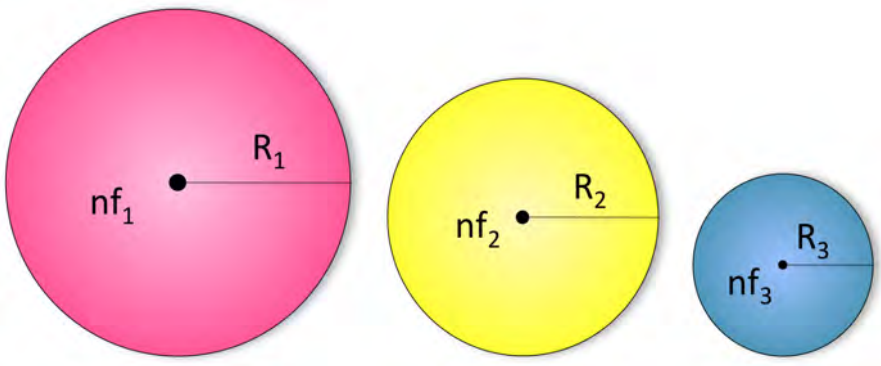

Figure 2.4: Species of UEGO.

defines sub-domains of the search space. As the algorithm progresses, the search process can be directed towards smaller regions by creating new sets of non-overlapping species defining smaller sub-domains. This process is a kind of cooling method similar to simulated annealing [149]. A particular species is not a fixed part of the search domain; it can move through space as the search proceeds. Additionally, UEGO is a hybrid algorithm that introduces a local optimizer into the evolution process [27, 155]. In this way, at every generation, UEGO performs a local optimizer operation on each species, and these locally optimal solutions replace the caller species. Notice that any single step taken by the optimizer in a given species is shorter than the radius of the given species.

UEGO is abstract in the sense that the 'species-management' and the cooling mechanism have been logically separated from the actual local optimization algorithm. Therefore it is possible to implement any kind of optimizer to work inside a species. For the endogenous demand problem considered in this chapter, a Weiszfeld-like algorithm has been developed (see Subsection 2.3.2). Additionally, the parameters that control UEGO have to be tuned to this new problem (see Subsection 2.3.3). The reader is referred to [135, 136] for a more detailed description of the UEGO algorithm.

2.3.2 Local optimizer

The local optimizer presented in [61] (see also [135]) for solving the corresponding fixed demand case is a steepest descent type method which takes discrete steps along the search paths and, in the best case, converges to a local optimum. The method sets the derivatives of the objective function to zero and the next iterate is obtained by implicitly solving the resulting equations. In Location Science, these types of methods are known as Weiszfeld-like methods, in honour of E. Weiszfeld, who first proposed that strategy in 1937 [157].

A similar algorithm can be devised for the variable demand model, although the mathematical development is more complicated. The details are given next. Whereas for the fixed demand model the corresponding Weiszfeld-like algorithm was able to obtain good solutions when used within a multi-start strategy (see [135]), this is no longer true for the variable demand model. Even when the Weiszfeld-like algorithm is started from 10000 different initial points, the best result obtained is often far from

the optimal solution. For the problem with variable demand, the objective function is steep and not very smooth. This causes a gradient-like local search not to converge.

However, the local search procedure helps to find the global optimum when used within the framework of UEGO, as the computational studies will show.

In the following, the market share captured by the chain can be rewritten as

$$M(nf) = \sum_{i=1}^{i_{\max}} w_i(U_i) \left(1 - \frac{\sum_{j=k+1}^{j_{\max}} u_{i,j}}{u_{i,nf} + \sum_{j=1}^{j_{\max}} u_{i,j}}\right),$$

and setting

$$r_i = \sum_{j=1}^{j_{\max}} u_{i,j}, \quad pu_i = \sum_{j=k+1}^{j_{\max}} u_{i,j},$$

we finally have

$$M(nf) = \sum_{i=1}^{i_{\max}} w_i(U_i) \left(1 - \frac{pu_i}{u_{i,nf} + r_i}\right) = \sum_{i=1}^{i_{\max}} \left(w_i(U_i) - \frac{w_i(U_i) g_i(d_i(z)) pu_i}{\gamma_i \alpha + r_i g_i(d_i(z))}\right).$$

Notice that U_i is a function of nf, with $U_i = U_i(nf) = u_{i,nf} + \sum_{j=1}^{j_{\max}} u_{i,j} = \dfrac{\gamma_i \alpha}{g_i(d_i(z))} + r_i$.

A necessary condition for a vector (z^*, α^*) to be a local or global maximum of $\Pi(z, \alpha) = F(M(z, \alpha)) - \sum_{i=1}^{i_{\max}} \Phi_i(d_i(z)) - G^b(\alpha)$ is that the partial derivatives of Π at that point must vanish.

With regard to the first variable, x, we have that

$$\frac{\partial \Pi}{\partial x} = 0 \Longleftrightarrow \frac{\mathrm{d}F}{\mathrm{d}M} \cdot \frac{\partial M}{\partial x} - \sum_{i=1}^{i_{\max}} \frac{\partial \Phi_i}{\partial x} = 0.$$

On the other hand,

$$\frac{\partial M}{\partial x} = \sum_{i=1}^{i_{\max}} \frac{\mathrm{d}M}{\mathrm{d}d_i(z)} \cdot \frac{\partial d_i(z)}{\partial x}$$

and

$$\frac{\partial \Phi_i}{\partial x} = \frac{\mathrm{d}\Phi_i}{\mathrm{d}d_i(z)} \frac{\partial d_i(z)}{\partial x}.$$

Hence,

$$\frac{\partial \Pi}{\partial x} = 0 \Longleftrightarrow \frac{\mathrm{d}F}{\mathrm{d}M} \sum_{i=1}^{i_{\max}} \frac{\mathrm{d}M}{\mathrm{d}d_i(z)} \cdot \frac{\partial d_i(z)}{\partial x} - \sum_{i=1}^{i_{\max}} \frac{\mathrm{d}\Phi_i}{\mathrm{d}d_i(z)} \frac{\partial d_i(z)}{\partial x} = 0.$$

If we denote

$$H_i(z,\alpha) = \frac{\mathrm{d}F}{\mathrm{d}M} \frac{\mathrm{d}M}{\mathrm{d}d_i(z)} - \frac{\mathrm{d}\Phi_i}{\mathrm{d}d_i(z)} = \frac{\mathrm{d}\Pi}{\mathrm{d}d_i(z)}$$

then

$$\frac{\partial \Pi}{\partial x} = 0 \Longleftrightarrow \sum_{i=1}^{i_{\max}} H_i(z,\alpha) \frac{\partial d_i(z)}{\partial x} = 0.$$

It is necessary to compute $\frac{\mathrm{d}F}{\mathrm{d}M}$, $\frac{\mathrm{d}M}{\mathrm{d}d_i(z)}$ and $\frac{\mathrm{d}\Phi_i}{\mathrm{d}d_i(z)}$.

If we assume that $F(M(z,\alpha)) = s \cdot M(z,\alpha)$ then $\frac{\mathrm{d}F}{\mathrm{d}M} = s$.

If we assume that $\Phi_i(d_i(z)) = Aver_{A_i}(w_i(U_i))/((d_i(z))^{\phi_{i,0}} + \phi_{i,1})$ then

$$\frac{\mathrm{d}\Phi_i}{\mathrm{d}d_i(z)} = -\frac{Aver_{A_i}(w_i(U_i))(d_i(z))^{\phi_{i,0}-1}}{((d_i(z))^{\phi_{i,0}} + \phi_{i,1})^2}$$

Taking into account that

$$\frac{\mathrm{d}w_i(U_i)}{\mathrm{d}d_i(z)} = incr_i \cdot \frac{\mathrm{d}e_i}{\mathrm{d}U_i} \cdot \frac{\mathrm{d}U_i}{\mathrm{d}d_i(z)} = \frac{-\alpha\gamma_i incr_i}{(g_i(d_i(z)))^2} \cdot \frac{\mathrm{d}e_i}{\mathrm{d}U_i} \frac{\mathrm{d}g_i}{\mathrm{d}d_i(z)}$$

then,

$$\frac{\mathrm{d}M}{\mathrm{d}d_i(z)} = \frac{\mathrm{d}\left(w_i(U_i) - \frac{w_i(U_i) g_i(d_i(z)) pu_i}{\gamma_i\alpha + r_i g_i(d_i(z))}\right)}{\mathrm{d}d_i(z)} = \frac{-\alpha\gamma_i incr_i}{(g_i(d_i(z)))^2} \cdot \frac{\mathrm{d}e_i}{\mathrm{d}U_i} \frac{\mathrm{d}g_i}{\mathrm{d}d_i(z)} -$$

$$\frac{\frac{-\alpha^2\gamma_i^2 incr_i pu_i}{g_i(d_i(z))} \cdot \frac{\mathrm{d}e_i}{\mathrm{d}U_i} \frac{\mathrm{d}g_i}{\mathrm{d}d_i(z)} - \gamma_i\alpha \cdot incr_i \cdot r_i \frac{\mathrm{d}e_i}{\mathrm{d}U_i} \frac{\mathrm{d}g_i}{\mathrm{d}d_i(z)} + w_i(U_i) \frac{\mathrm{d}g_i}{\mathrm{d}d_i(z)} pu_i\gamma_i\alpha}{(\gamma_i\alpha + g_i(d_i(z))r_i)^2}.$$

In particular, if we assume that $g_i(d_i(z)) = (d_i(z))^2$ (as is most commonly done in literature) then $\frac{\mathrm{d}g_i}{\mathrm{d}d_i(z)} = 2d_i(z)$ and the previous expression becomes

$$\frac{\mathrm{d}M}{\mathrm{d}d_i(z)} = \frac{-\alpha\gamma_i incr_i \cdot 2}{(d_i(z)))^3} \cdot \frac{\mathrm{d}e_i}{\mathrm{d}U_i} -$$

$$\frac{\frac{-\alpha^2\gamma_i^2 incr_i pu_i}{d_i(z)} \cdot \frac{\mathrm{d}e_i}{\mathrm{d}U_i} \cdot 2 - \gamma_i\alpha \cdot incr_i \cdot r_i \frac{\mathrm{d}e_i}{\mathrm{d}U_i} 2d_i(z) + w_i(U_i)2d_i(z)pu_i\gamma_i\alpha}{(\gamma_i\alpha + (d_i(z))^2 r_i)^2}.$$

Also notice that if $e_i(U_i) = e_{i_1}(U_i) = c_i U_i$ then $\frac{\mathrm{d}e_i}{\mathrm{d}U_i} = c_i$.

If $d_i(z)$ is a distance function such that

$$\frac{\partial d_i(z)}{\partial x} = xA_{i,1}(z) - B_{i,1}(z),$$

where $A_{i,1}(z)$ and $B_{i,1}(z)$ are functions of z, then

$$\frac{\partial \Pi}{\partial x} = 0 \Longleftrightarrow x = \frac{\sum_{i=1}^{i_{\max}} H_i(z,\alpha)B_{i,1}(z)}{\sum_{i=1}^{i_{\max}} H_i(z,\alpha)A_{i,1}(z)}.$$

Analogously, if $\frac{\partial d_i(z)}{\partial y} = yA_{i,2}(z) - B_{i,2}(x)$, then

$$\frac{\partial \Pi}{\partial y} = 0 \Longleftrightarrow y = \frac{\sum_{i=1}^{i_{\max}} H_i(z,\alpha)B_{i,2}(z)}{\sum_{i=1}^{i_{\max}} H_i(z,\alpha)A_{i,2}(z)}.$$

Finally,

$$\frac{\partial \Pi}{\partial \alpha} = 0 \Longleftrightarrow \frac{\mathrm{d}F}{\mathrm{d}M} \cdot \frac{\partial M}{\partial \alpha} - \frac{\mathrm{d}G^b}{\mathrm{d}\alpha} = 0.$$

For this last expression we need to compute $\frac{\mathrm{d}F}{\mathrm{d}M}$, $\frac{\partial M}{\partial \alpha}$ and $\frac{\mathrm{d}G^b}{\mathrm{d}\alpha}$.

If we assume that $F(M(z,\alpha)) = s \cdot M(z,\alpha)$, then $\frac{\mathrm{d}F}{\mathrm{d}M} = s$.

If we assume that $G^b(\alpha) = e^{\frac{\alpha}{\alpha_0}+\alpha_1} - e^{\alpha_1}$ then

$$\frac{\mathrm{d}G^b}{\mathrm{d}\alpha} = \frac{1}{\alpha_0} e^{\frac{\alpha}{\alpha_0}+\alpha_1}$$

Taking into account that

$$\frac{\partial w_i}{\partial \alpha} = incr_i \frac{\mathrm{d}e_i}{\mathrm{d}U_i}\frac{\mathrm{d}U_i}{\mathrm{d}\alpha} = incr_i \frac{\mathrm{d}e_i}{\mathrm{d}U_i}\frac{\gamma_i}{g_i(d_i(z))},$$

then

$$\frac{\partial M}{\partial \alpha} =$$

$$\sum_{i=1}^{i_{\max}} \left(incr_i \frac{\mathrm{d}e_i}{\mathrm{d}U_i}\frac{\gamma_i}{g_i(d_i(z))} - \frac{incr_i \frac{\mathrm{d}e_i}{\mathrm{d}U_i}\gamma_i pu_i(\gamma_i\alpha + g_i(d_i(z))r_i) - w_i(U_i)g_i(d_i(z))pu_i\gamma_i}{(\gamma_i\alpha + g_i(d_i(z))r_i)^2} \right)$$

Notice that if we fix $z = (x, y)$, then the equation $\dfrac{\partial \Pi}{\partial \alpha} = 0$ has just one variable, α. Thus we could solve it by using any algorithm for solving equations of a single variable, for instance, the Newton or the Secant methods.

Among the distance functions that satisfy the conditions

$$\frac{\partial d_i(z)}{\partial x} = xA_{i,1}(z) - B_{i,1}(z), \frac{\partial d_i(z)}{\partial y} = yA_{i,2}(z) - B_{i,2}(z),$$

we have the inflated Euclidean distance or its rescaled version, the l_{2b} norm, given by

$$d_i(z) = \sqrt{b_1(x - locd_{i1})^2 + b_2(y - locd_{i2})^2},$$

where $b_1, b_2 > 0$ are given parameters (see [58]). In this case

$$A_{i,1} = \frac{b_1}{d_i(z)}, \quad A_{i,2} = \frac{b_2}{d_i(z)}$$
$$B_{i,1} = \frac{locd_{i1}b_1}{d_i(z)} \quad B_{i,2} = \frac{locd_{i2}b_2}{d_i(z)}$$

Thus, the following Weiszfeld-like algorithm can be constructed.

Algorithm 4: WLMv (Weiszfeld-like algorithm)

1: Set $ic = 0$
2: Set $nf^{(0)} = (x^{(0)}, y^{(0)}, \alpha^{(0)})$.
3: **while** termination criteria are not satisfied **do**
4: Update $nf^{(ic+1)} = (x^{(ic+1)}, y^{(ic+1)}, \alpha^{(ic+1)})$
5: **if** $nf^{(ic+1)}$ is unfeasible **then**
6: $nf^{(ic+1)} \in [nf^{(ic)}, nf^{(ic+1)}] \cap \partial FR$
7: $ic = ic + 1$

The values $x^{(ic+1)}$ and $y^{(ic+1)}$ of Algorithm 4 are obtained as follows:

$$x^{(ic+1)} = \frac{\sum_{i=1}^{i_{\max}} H_i(x^{(ic)}, y^{(ic)}, \alpha^{(ic)}) B_{i,1}(x^{(ic)}, y^{(ic)})}{\sum_{i=1}^{i_{\max}} H_i(x^{(ic)}, y^{(ic)}, \alpha^{(ic)}) A_{i,1}(x^{(ic)}, y^{(ic)})}$$

$$y^{(ic+1)} = \frac{\sum_{i=1}^{i_{\max}} H_i(x^{(ic)}, y^{(ic)}, \alpha^{(ic)}) B_{i,2}(x^{(ic)}, y^{(ic)})}{\sum_{i=1}^{i_{\max}} H_i(x^{(ic)}, y^{(ic)}, \alpha^{(ic)}) A_{i,2}(x^{(ic)}, y^{(ic)})}$$

and $\alpha^{(ic+1)}$ as a solution of the equation

$$\frac{\partial \Pi}{\partial \alpha}(x^{(ic+1)}, y^{(ic+1)}, \alpha^{(ic)}) = 0$$

According to steps 5 and 6 of Algorithm 4, if f $nf^{(ic+1)}$ is unfeasible then $nf^{(ic+1)}$ is set equal to a point in the segment $[(x^{(ic)}, y^{(ic)}, \alpha^{(ic)}), (x^{(ic+1)}, y^{(ic+1)}, \alpha^{(ic+1)})]$ which is on the border of the feasible region. Also notice that when solving the equation for $\alpha^{(ic+1)}$ in Step 4 of Algorithm 4, we do not need to obtain $\alpha^{(ic+1)}$ exactly: an approximation is sufficient. So, when using a solution procedure for the equation, we do not need to wait for the convergence of the procedure: a few iterations may be enough. Similarly, in Step 6, it is not necessary to obtain an exact boundary point; any close but feasible point will do.

Several termination criteria have been studied for the WLMv algorithm (Step 3, Algorithm 4), although only two of them have been considered at the end. The first one is based on the distance between the consecutive iterative vectors $(x^{(ic)}, y^{(ic)}, \alpha^{(ic)})$ and $(x^{(ic+1)}, y^{(ic+1)}, \alpha^{(ic+1)})$. Different functions can be used to measure the closeness of two consecutive vectors. For instance, one of them may be to stop the algorithm if $\|(x^{(ic)}, y^{(ic)}) - (x^{(ic+1)}, y^{(ic+1)})\|_2 < \epsilon_1$ and $|\alpha^{(ic)} - \alpha^{(ic+1)}| < \epsilon_2$, for given tolerances $\epsilon_1, \epsilon_2 > 0$. The second stopping criterion requires setting a maximum number of iterations ic_{max} because the convergence of the algorithm cannot be guaranteed. When a termination criteria is satisfied the vector $(x^{(ic+1)}, y^{(ic+1)}, \alpha^{(ic+1)})$ is accepted as a potential local maximum.

2.3.3 Tuning UEGO

In [135], it was found that a good parameter setting for UEGO to solve the exogenous demand problem was to set the maximum number of function evaluations allowed for the whole optimization process to $FE_{\max} = 10^6$, the maximum number of species kept during the execution of the algorithm to $L_{\max} = 150$, the radius of the smallest species to $R_{t_{\max}} = 0.005$ and the number of levels or cooling stages (the number of times that the radii of the species are reduced) to $t_{\max} = 30$.

However, for solving the endogenous demand problem, the parameter setting has been modified by increasing the maximum number of species to $L_{\max} = 350$ and the radius of the smallest species to $R_{t_{\max}} = 0.05$. The remaining parameters maintain the same value. These modifications have been introduced to carry out a deeper search (a denser covering of the landscape) and to reduce the computational cost, respectively. Remember that this is a hard-to-solve global optimization problem with many local optima. Therefore, the larger the number of species, the deeper the search of the global optimum, but also the higher the computational effort. To reduce the execution times while maintaining the quality in the solutions, the radius $R_{t_{\max}}$ has been increased. The interested reader is referred to [125] for an in-detail description of the UEGO parameters and their effects on obtaining a robust parameter setting.

2.3.4 Computational studies

All the computational results in this section have been obtained under Linux on an AMD Athlon(tm) 64 X2 with 2.2GHz CPU and 2GB memory. UEGO has been implemented in C++. As for the interval B&B method (iB&B), the implementation by B. Tóth used in [61] has been employed, which uses the interval arithmetic of the PROFIL/BIAS library [105] and the automatic differentiation of the C++ Toolbox library [85].

In order to evaluate the performance of UEGO, a representative set of location problems has been generated, varying the number $i_{\max}$ of demand points, the number $j_{\max}$ of existing facilities and the number k of those facilities belonging to the chain. In particular, both small ($i_{\max} \leq 200$) as well as large size problems ($i_{\max} \geq 500$) have been generated. The settings $(i_{\max}, j_{\max}, k)$ employed in the problems can be seen in Table 2.1 (Page 46).

For every setting with $i_{\max} \leq 200$, 10 problems were generated by randomly choosing the parameters of the problems uniformly within the intervals presented in the Appendix. Similarly, for every setting with $i_{\max} \geq 500$, 5 problems were generated. The search space for every problem was

$$z \in FR, \quad \alpha \in [0.5, 5].$$

The sets used for FR can be found in Table 2.1. For the Weiszfeld-like method, the tolerances were set to $\epsilon_1 = \epsilon_2 = 10^{-2}$ and the maximum iteration counter $ic_{\max}$ to 400, see Subsection 2.3.2.

Solving small problems

Since UEGO is a heuristic, different runs may provide different solutions. To take this effect into account, UEGO has been run five times for each problem. In each run, we obtain the approximate optimal value vP_{ap}, the point mP_{ap} at which that value is attained and the CPU time employed by the algorithm. With this information we investigate whether UEGO has successfully found the optimal solution. To this aim, we check whether both vP_{ap} and mP_{ap} are included in the corresponding intervals provided by the exact interval branch-and-bound algorithm, see [61, 153], which was run with

$i_{\max} = 50$				$i_{\max} = 100$				$i_{\max} = 200$			
$j_{\max}$	k	Π	I_{UEGO}	$j_{\max}$	k	Π	I_{UEGO}	$j_{\max}$	k	Π	I_{UEGO}
2	0	1896.94	30.27	2	0	1500.04	38.02	2	0	501.26	76.72
	1	2647.48	30.88		1	1905.06	37.21		1	571.86	77.11
5	0	2878.26	29.70	5	0	1979.66	33.86	10	0	1.03	80.93
	1	3365.72	37.62		1	2254.91	35.23		2	116.13	81.30
	2	4084.44	34.98		2	2766.80	29.18		4	529.08	82.83
10	0	36.12	21.52	10	0	22.94	44.74	15	0	424.40	80.12
	2	546.41	21.07		2	437.57	35.64		5	959.05	80.95
	4	1412.11	21.42		4	1308.44	29.33		10	2045.98	82.56

Table 2.4: Average results for small problems.

a tolerance of 10^{-4}. Note that those intervals (whose maximum width can be at most that of the tolerance) contain any optimal solution to the problem. It is important to highlight that UEGO has a 100% rate of success, i.e. it has always found, in all the runs and in all the problems, an approximation of the global optimum solution which is included in the solution list of 3-dimensional intervals offered by the iB&B method (and of course, its optimal value is included in the corresponding interval offered by the interval method).

And as expected, UEGO has used less computational time than the iB&B algorithm. Table 2.4 shows average values of the 10 problems solved for each setting $(i_{\max}, j_{\max}, k)$. In particular, the average objective value (Π) and the average time improvement (I_{UEGO}) obtained by UEGO, as compared to the iB&B, in percentage, are shown. As we can observe, such improvement increases with the computational load of the problem. In fact, the improvement obtained by UEGO increases, in average, from 28.43% when $i_{\max} = 50$ to 80.32% when $i_{\max} = 200$. Hence, we can conclude that UEGO is a reliable method (it has always obtained the optimal solution) and much faster than the iB&B method (but notice that the interval method is an exact method which always finds the optimal solution with guarantee).

Solving larger problems

In this subsection, the behavior of UEGO when the number of demand points is larger than or equal to $i_{\max} = 500$ is studied. In this case, the iB&B method is not able to

solve the problems (the computer runs out of memory). For this reason, the percentage of success at finding the optimal solution is not computed, since the optimal solution is not known with guarantee. Instead, UEGO has been run five times for each problem and different measures have been computed. In particular, the average time ($Time$) in the five runs (in seconds), the minimum (Min), the average (Av) and the maximum (Max) objective value in the five runs, and finally the standard deviation (Dev), have been computed. Table 2.5 summarizes those results for each setting $(i_{\max}, j_{\max}, k)$. The given values correspond to the mean values for all the problems with the same setting. As we can see, the differences between the values in columns Min and Max are always negligible (also see column Dev), which gives an idea of the robustness of the algorithm. In fact, the Euclidean distance between any pair of solutions provided by UEGO, in the five runs was always less than 10^{-12}, which shows that the algorithm always finds the same solution in all the runs. UEGO can solve the most complicated problems, with $(i_{\max} = 1000, j_{\max} = 50, k = 0)$ settings, in less than 33 minutes of CPU time.

				Objective Function			
$i_{\max}$	$j_{\max}$	k	$Time$	Min	Av	Max	Dev
500	2	0	528.188	180.63510	180.63513	180.63515	0.00002
		1	524.558	186.22636	186.22636	186.22636	0.00000
	15	0	513.384	185.95095	185.95095	185.95095	0.00000
		5	483.376	212.96355	212.96355	212.96355	0.00000
		10	581.304	224.25499	224.25499	224.25499	0.00000
	25	0	547.184	190.70291	190.70292	190.70293	0.00001
		7	587.638	221.74127	221.74128	221.74129	0.00001
		15	597.838	258.23334	258.23334	258.23334	0.00000
1000	5	0	1805.698	186.21116	186.21117	186.21118	0.00001
		1	1840.938	186.90243	186.90243	186.90243	0.00000
		2	1843.256	187.48302	187.48302	187.48302	0.00000
	25	0	1734.526	185.99122	185.99122	185.99122	0.00000
		7	1697.524	191.93223	191.93223	191.93223	0.00000
		15	1737.840	199.43877	199.43877	199.43877	0.00000
	50	0	1934.122	185.09211	185.09212	185.09213	0.00001
		15	1692.750	201.68740	201.68740	201.68740	0.00000
		30	1697.096	223.66563	223.66563	223.66563	0.00000

Table 2.5: Average results for large problems.

2.4 Sensitivity analysis

The endogenous demand location model is a highly nonlinear optimization problem, whose behavior and parameters may be difficult to understand well. Therefore, it could be interesting to know how stable the solution is with respect to those parameters. This calls for a sensitivity analysis of the optimal solution in terms of changes in the parameters. This section contributes to this study, in particular, by researching the changes in optimal design/location when the parameters related to the demand and the quality change. Changes in the other parameters will be left for future research. As for the fixed demand location problem, the reader is referred to [154], where a complete sensitivity analysis of that model is presented.

To carry out our studies, the iB&B method with a tolerance of 10^{-6} has been used.

2.4.1 On the variability of the demand

In this subsection, the stability of the solution when the average demand $\widehat{w}_i$ at $locd_i$ $(i = 1, .., i_{\max})$ varies is investigated. For the study at hand, a total of 48 problems have been selected, i.e., 2 problems for every setting with $i_{\max} \leq 200$ in Table 2.1 (Page 46). Sixteen versions of each problem have been solved, applying a variation, in percentage, of $\pm 0.25, \pm 0.5, \pm 0.75, \pm 1, \pm 2, \pm 3, \pm 4, \pm 5$ to every $\widehat{w}_i$ $(i = 1, .., i_{\max})$.

Table 2.6 summarizes the solutions obtained by the iB&B method for the instance with setting $(i_{\max} = 50, j_{\max} = 5, k = 2)$ already used in Section 2.2. In particular, the corresponding global maximizer point(s) mP_{exact} (actually, a 3-dimensional box containing the small 3-dimensional boxes in the solution list where any maximizer point must lie) and its global optimal value vP_{exact} (actually, a very narrow interval $[\underline{vP}_{exact}, \overline{vP}_{exact}]$ containing it), are shown. In the table, the notation $67.505[198, 218]$, for instance, stands for $[67.505198, 67.505218]$. As can be observed, the optimal solution point usually changes in a progressive way as the demand increases. However, sometimes there is a jump in the optimal solution point. See, for instance, the solutions when the average demand is modified from -2% to -1%, in Table 2.6. Another smaller jump can be seen when the average demand is modified from $+2\%$ to $+3\%$. These jumps are not uncommon, and can be observed in 13 out of the 48 instances of our study.

	mP_{exact}			
w_i	x	y	α	$[\underline{vP}_{exact}, \overline{vP}_{exact}]$
-5%	0.000[000,008]	0.823[581,583]	0.500[000,000]	48.614[065,070]
-4%	0.000[000,008]	0.817[895,897]	0.500[000,000]	49.954[607,612]
-3%	0.000[000,007]	0.812[213,215]	0.500[000,000]	51.309[837,841]
-2%	0.000[000,007]	0.806[535,538]	0.500[000,000]	52.679[688,693]
-1%	0.732[494,497]	5.889[163,165]	2.033[438,452]	54.129[522,567]
-0.75%	0.732[261,263]	5.887[340,342]	2.053[345,358]	54.614[017,063]
-0.50%	0.732[027,031]	5.885[519,520]	2.073[249,264]	55.101[778,835]
-0.25%	0.731[794,797]	5.883[698.699]	2.093[151,163]	55.592[795,841]
0.00%	0.731[561,563]	5.881[879,879]	2.113[054,064]	56.087[634,660]
+0.25%	0.731[327,329]	5.880[061,062]	2.132[941,953]	56.584[672,719]
+0.50%	0.731[094,097]	5.878[244,246]	2.152[830,842]	57.085[547,584]
+0.75%	0.730[860,863]	5.876[429,249]	2.172[716,726]	57.589[729,761]
+1%	0.730[627,630]	5.874[615,615]	2.192[592,598]	58.097[249,275]
+2%	0.729[693,697]	5.867[370,372]	2.272[097,111]	60.160[891,951]
+3%	1.095[527,530]	5.747[775,775]	2.453[591,599]	62.669[857,870]
+4%	1.056[957,960]	5.742[340,341]	2.547[668,676]	65.084[199,218]
+5%	1.026[317,319]	5.738[266,266]	2.635[716,724]	67.505[198,218]

Table 2.6: Variation of the optimal solution for a problem with setting ($i_{\max} = 50, j_{\max} = 5, k = 2$).

Regarding the objective function value, it increases as the demand also increases. It happened in most of the problems (44 out of 48).

To have a general overview of the results, the variation in the profit between the cases -5% and $+5\%$ has also been computed, considering the midpoint in $[\underline{vP}_{exact}, \overline{vP}_{exact}]$. In Table 2.7, we summarize those results, and we give the average of the variation, in percentage, for each value of $i_{\max}$ and $j_{\max}$.

As we can see, although the absolute variation in the average demand is just 10%, the variation in the objective function value is greater, extending from 13% to 33%. Hence, it is important to have good estimations of the average values of the demand if we want our location model to produce reliable results.

2.4.2 On the interval for the quality

The quality of the new facility clearly affects the profit obtained by the chain. However, since the quality of the existing facilities is known in advance, one may wonder whether

$i_{\max}$	$j_{\max}$	Av(Var)(%)
50	2	23.158
	5	18.967
	10	17.924
100	2	18.635
	5	20.727
	10	13.232
200	2	29.032
	10	33.056
	15	26.730

Table 2.7: Average values for profit variation.

it is possible to select the quality of the new facility in advance, taking the quality of the existing facilities into account, without having to consider the quality as one of the variables of the problem or, at least, select a range within which the search for the quality of the new facility can be restricted.

We have tried four different scenarios, where the interval $[\alpha^{\min}, \alpha^{\max}]$ has been set to:

1. $[\max\{a_{i,j}/\gamma_i : i = 1, \ldots, i_{\max},\ j = 1 \ldots, j_{\max}\}, \max\{a_{i,j}/\gamma_i : i = 1, \ldots, i_{\max},\ j = 1 \ldots, j_{\max}\} + 4]$, so that the utility of the new facility will be the highest one in the region.

2. $[\max\{a_{i,j}/\gamma_i : i = 1, \ldots, i_{\max},\ j = 1 \ldots, k\}, \max\{a_{i,j}/\gamma_i : i = 1, \ldots, i_{\max},\ j = 1 \ldots, k\} + 4]$, so that the utility of the new facility will be higher than that of any of the facilities owned by the chain.

3. $[\max\{a_{i,j}/\gamma_i : i = 1, \ldots, i_{\max},\ j = k+1 \ldots, j_{\max}\}, \max\{a_{i,j}/\gamma_i : i = 1, \ldots, i_{\max}, j = k+1 \ldots, j_{\max}\} + 4]$, so that the utility of the new facility will be higher than that of any of the facilities of its competitors.

4. $[0.5, 1.5 \cdot \max\{a_{i,j}/\gamma_i : i = 1, \ldots, i_{\max},\ j = 1 \ldots, j_{\max}\} + 4]$, so that the utility of the new facility will be selected within an interval which contains all the possible utilities of the existing facilities and also greater utilities.

For this experiment, one problem for every setting with $i_{\max} \leq 200$ in Table 2.1

(Page 46) has been selected, and the difference in profit obtained with each scenario as compared to the original problem, where α belongs to $[0.5, 5]$, has been studied.

The results were a bit surprising. In the first scenario, only in 7 out of the 24 problems did the profit increase. In the second scenario, only in 11 problems was a better profit obtained, and in the third and fourth scenarios the number of problems with a better profit were 7 and 9, respectively. Furthermore, it is important to mention that for the same subset of 7 problems in scenario 1 where a better objective function value was obtained, a better profit was also obtained in the other scenarios. For the remaining problems in scenarios 1, 3 and 4 there was a decrease in the objective function value. In these cases, the cost of achieving the quality is comparatively greater than the income that it may produce and, as a result, there is a decrement in the profit. This is because the optimal value for α when looking for it in the interval $[0, +\infty]$ is smaller than the lower bound of the range interval of such scenarios. Regarding scenario 2, the same profit has been obtained for the remaining 13 problems. For those cases, the optimal value for α is less than 5.0, so the optimal solution remains the same and enlarging the interval where the quality of the new facility must lie does not produce changes.

Hence, it can be concluded that setting α to higher values does not necessarily imply an increment in the profit obtained by the chain. The suitable value for α depends on the particular instance we consider. Therefore, the best option is to solve the problem with a range for α including both lower and higher values, as in the fourth scenario.

2.4.3 On the customers' sensitivity

Since the attraction that customers feel for the facilities depends on both the quality of the facility and the distance to the facility, one may wonder whether it is possible to adjust the parameters of the model assuming a certain situation, e.g., the case where customers are more sensitive to the quality of new facility than to its distance, or the opposite case, where customers are more sensitive to the distance rather than to the quality.

The attraction that the customers at $locd_i$ feel for the new facility is given by $u_{i,nf} = \gamma_i \alpha / g_i(d_i(z))$. Hence, if the range for α is wider than that of $g_i(d_i(z))$, then the sensitivity to the quality will be greater than to the distance, and vice versa.

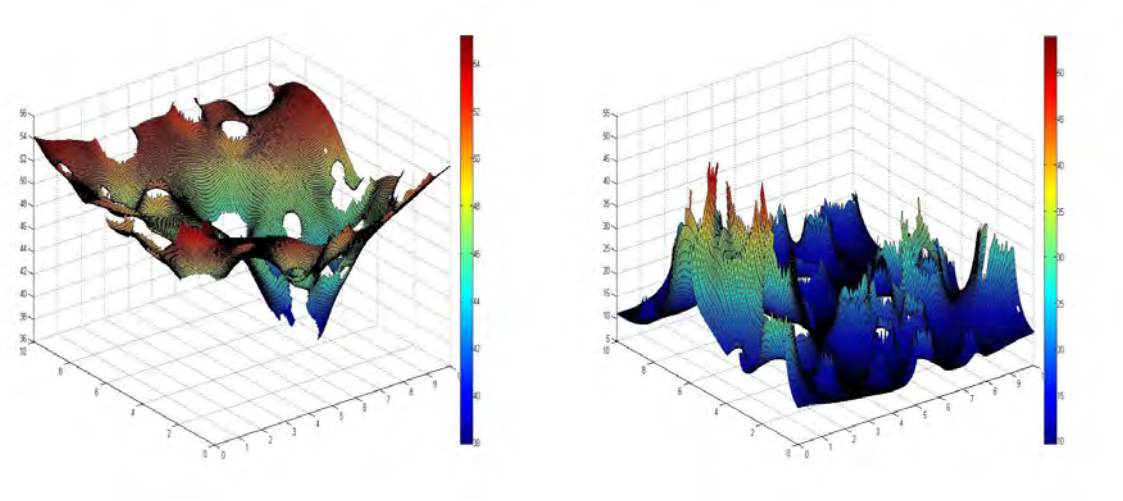

Figure 2.5: Objective function of the problem with setting ($i_{\max} = 50, j_{\max} = 5, k = 2$) and $FR = ([0, 10, [0, 10])$, projected in the 2-dimensional location space, when $\alpha, a_{i,j} \in [0.5, 5]$, with (a) $\alpha = 0.5$ on the left (b) $\alpha = 2.75$ on the right.

To corroborate this fact, we have compared the problems with settings ($i_{\max} = 71, j_{\max} = 5, k = 0$) and ($i_{\max} = 50, j_{\max} = 5, k = 2$) already used in Section 2.2, in two different situations. Since the region of the plane where the new facility can be located is $FR = ([0, 10], [0, 10])$ and $g_i(d_i(z)) = d_i(z)^2$ for all i, using the Euclidean distance the range for $g_i(d_i(z))$ is (within) the interval $[0, 200]$. We have first solved the problems assuming that both α and $a_{i,j}$ lie in the interval $[0.5, 5]$. In Figure 2.5, we give the graph of the objective function on the location domain for the problem with setting ($i_{\max} = 50, j_{\max} = 5, k = 2$) for two different values of α, 0.5 and 2.75. Second, we have solved the same problems but multiplying $a_{i,j}$ by 20 and setting the interval for α to $[0.5, 100]$. In Figure 2.6, we give the graph of the objective function on the location domain for the same problem for two different values of α, 0.5 and 50.

Comparing figures 2.5a and 2.6a (notice that in both cases $\alpha = 0.5$) we can see that the width of the range of variability of the objective function value is quite similar, between 40 and 55 in Figure 2.5a and between 1289 and 1304 in Figure 2.6a. But although the absolute variability is similar, (around 15 in both cases), the relative variability in the first case (15/55) is much bigger than in the second one (15/1304). So, the smaller the range of the quality as compared to the distance, the more the objective varies with changes in the location, that is, the more the objective function

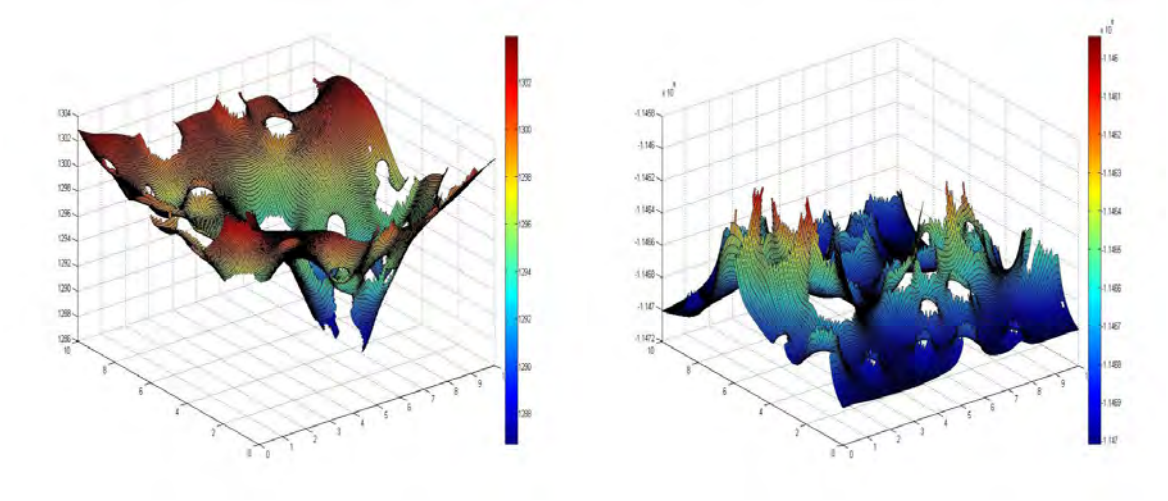

Figure 2.6: Objective function of the problem with setting ($i_{\max} = 50, j_{\max} = 10, k = 2$) and $FR = ([0, 10, [0, 10])$, projected in the 2-dimensional location space, when $\alpha \in [0.5, 100]$, with (a) $\alpha = 0.5$ on the left (b) $\alpha = 50$ on the right.

with the distance to the facility varies: customers are more sensitive to distance.

On the other hand, looking at figures 2.5a and 2.5b, we see that the range of variability of the objective function is very similar. This is because in both pictures the value of α is small as compared to the distance. However, in Figure 2.6, the quality of the new facility (and that of the existing ones) can vary much more (in a magnitude similar to the distance), that is why now the range of variability of the objective function is very different between the cases $\alpha = 0.5$ and $\alpha = 50$. But notice that the shapes of the graphs in figures 2.5b and 2.6b are very similar, which means that the location of the facility still plays a role in the problem, although in the second figure (when the quality is of the order of the distance) the influence is much smaller as compared to the first one.

The same conclusions could be inferred from the instance with setting ($i_{\max} = 71, j_{\max} = 5, k = 0$), although for this reason we have not included the corresponding figures.

2.4.4 The cost of the exogenous demand assumption

To measure how important taking the variability of demand into account may be, we have conducted a final study in which, for the 345 problems described in Subsection

	Loss				NumProb	NumProb
$i_{\max}$	Min	Av	Max	Dev	Loss $> 10\%$	Loss $> 25\%$
50	0.000	10.464	37.150	9.325	26	8
100	3.500	23.859	49.950	9.981	73	34
200	0.030	33.997	65.380	16.660	37	26
500	31.200	49.737	90.380	13.764	40	40
1000	17.140	54.513	99.590	17.898	45	44

Table 2.8: Loss in profit when assuming fixed demand.

2.3.4, we have calculated the loss in profit when fixed demand is assumed instead of variable demand. To this aim, we have first solved the corresponding location problem assuming variable demand, and then assuming fixed demand (similar to what is done in Section 2.2). Then we have evaluated the objective function of the problem with variable demand at the optimal solution point obtained for that problem when assuming fixed demand, and computed the difference between that value and the optimal value of the problem with variable demand, that is, the loss in profit caused by the fixed demand assumption.

Table 2.8 summarizes the results, grouped according to the number of demand points. We give the minimum, the average and the maximum loss in profit obtained for all the problems with the same $i_{\max}$, in percentage, as well as the standard deviation. Additionally, the number of problems whose loss in profit is larger than 10% and 25% is also shown.

As can be seen, for 221 out of 345 problems, the loss is larger than 10%, and in 152 out of 345 problems it is over 25%. This clearly indicates the usefulness of considering variable demand, since the gains that can be achieved are substantial.

2.5 Improving the efficiency of UEGO: UEGOf

In Subsection 2.3.2, a Weiszfeld-like algorithm (WLMv, see Algorithm 4) was proposed for dealing with the problem at hand. However, the method, in the best case, converges to a local optimum. Furthermore, it may not converge. That is why we set a maximum number of 400 iterations in Subsection 2.3.2 as a second stopping rule for WLMv. The computational studies showed that between 1.5% and 5% of the calls to the Weiszfeld-

$i_{\max}$	5000			10000		
$j_{\max}$	10	50	100	20	100	200
k	0,2,4	0,15,30	0,25,50	0,5,10	0,25,50	0,50,100
FR	$([0,50],[0,50])$					

Table 2.9: More settings of larger problems.

like algorithm in UEGO stopped with this rule.

After a deeper study of the behavior of the algorithm, we have found that sometimes a new iterate may have a worse (smaller) objective function value than its predecessor (perhaps because the step given by the algorithm is too long), and although in future iterations the objective function value may increase, the improvement does not usually reach the best objective function value previously found by the algorithm. Thus, we propose a new (third) stopping rule for the Weiszfeld-like algorithm: we stop it whenever the objective function value of an iterate is worse than the one of its predecessor. UEGO, with this modified local search, will be called UEGOf throughout the thesis.

In order to show that UEGOf works as UEGO, in the sense that they both obtain similar solutions, the same representative set of location problems as in Subsection 2.3.4 has been considered. In addition, the set of larger problems has been increased considering 5 problems per setting of Table 2.9. They have been solved in the Ben supercomputer, using a single core.

Since both UEGO and UEGOf are heuristic algorithms, different runs may provide different solutions. To take this effect into account, they have been run five times for each problem. At each run, we obtain the approximate optimal value $vPap$, the point $mPap$ at which that value is attained and the CPU time employed by the algorithm. With this information we investigate whether UEGOf has successfully found the optimal solution as we did with UEGO in Subsection 2.3.4, that is, for the small problems we check whether both $vPap$ and $mPap$ are included in the corresponding intervals provided by the interval branch-and-bound algorithm, which was run with a tolerance of 10^{-4}. The results confirm that UEGOf also has (like UEGO) a 100% rate of success, i.e., it has always found, in all the runs and in all the problems, an approximation of the global optimum solution which is included in the solution list of 3-dimensional boxes offered by iB&B.

		Time			% iB&B		% UEGO
$i_{\max}$	$j_{\max}$	iB&B	UEGO	UEGOf	I_{UEGO}	I_{UEGOf}	I_{UEGOf}
50	2	42.377	5.974	8.894	85.90	79.01	-48.86
	5	56.757	7.651	8.165	86.52	85.61	-6.71
	10	57.860	8.547	8.443	85.23	85.41	1.21
100	2	232.786	17.063	14.593	92.67	93.73	14.48
	5	254.252	20.500	15.039	91.94	94.08	26.64
	10	293.232	22.917	14.267	92.18	95.13	37.75
200	2	1320.607	36.510	28.821	97.24	97.82	21.06
	10	1491.524	33.246	28.755	97.77	98.07	13.51
	15	1473.295	32.471	28.341	97.80	98.08	12.72

Table 2.10: Average results for small problems.

In Table 2.10, we give the mean time (in seconds) employed by the different algorithms (columns 3 to 5), grouped for each combination $(i_{\max}, j_{\max})$, when solving the small problems. We also give the relative reduction in CPU time for each pair of algorithms (columns 6 to 8). As we can see, except for the easiest problems with setting $(i_{\max} = 50, j_{\max} = 2, 5)$, UEGOf is faster than UEGO, with reductions varying from 1% up to 37% in some cases.

On the other hand, iB&B is not able to solve the larger problems (the computer runs out of memory). Hence, we cannot compute the percentage of success of the heuristics in finding the optimal solution for them, since the optimal solution is not known with guarantee. However, so as to check the reliability of the heuristics, they have been run five times for each problem, and the average time (in seconds) in the five runs, the average objective function value (columns Π) and the maximum Euclidean distance between any pair of solutions in the five runs (columns $MaxDist$) have been computed. Noteworthy is that a value equal to zero in the $MaxDist$ measure indicates that the corresponding algorithm is able to find the same solution in all the runs (at least up to 3 decimal places of accuracy), which gives an idea about its robustness.

Table 2.11 summarizes the results for UEGO and UEGOf. The given results correspond to the mean values for all the problems grouped for each value of $i_{\max}$ and $j_{\max}$. As can be seen, the differences between any pair of solutions in the five runs are always negligible for both UEGO and UEGOf, which shows their robustness (see columns $MaxDist$). And of course, the solutions obtained by the algorithms is practi-

		UEGO			UEGOf			
$i_{\max}$	$j_{\max}$	$Time$	Π	$MaxDist$	$Time$	Π	$MaxDist$	I_{UEGOf}
500	2	429.403	12.678	0.000	292.183	12.678	0.000	31.96
	15	446.686	24.262	0.000	285.469	24.262	0.000	36.09
	25	406.911	28.768	0.000	281.244	28.768	0.000	30.88
1000	5	959.340	0.887	0.000	345.629	0.886	0.000	63.97
	25	1036.678	-6.056	0.000	350.513	-6.056	0.000	66.19
	50	1049.559	14.900	0.000	361.128	14.901	0.000	65.59
5000	10	-	-	-	1750.307	-43.214	0.000	-
	50	-	-	-	1737.301	-30.972	0.000	-
	100	-	-	-	1694.765	-20.803	0.000	-
10000	20	-	-	-	2908.233	-28.582	0.000	-
	100	-	-	-	2922.019	-21.677	0.000	-
	200	-	-	-	2910.959	-13.637	0.000	-

Table 2.11: Average results for larger problems.

cally the same (see columns Π). The slight differences in objective function value are due to the high steepness of the objective function value around the optimal solution, which makes that very close feasible points may have very different objective function values. Furthermore, as with the small instances, it can be seen that UEGOf is much faster than UEGO, with reductions varying from 30% to 66% (see I_{UEGOf} column).

Remarkably, UEGO has not been able to solve the largest problems, i.e. those with setting $i_{\max} \geq 5000$. On the contrary, UEGOf can tackle those problems without difficulty, since the deployment of the new optimization method not only reduces the computing time, but also the memory resources.

2.6 High performance computing

As we have seen, the CPU time needed by UEGOf to solve the largest problems is not negligible (around 50 minutes). Despite the fact that the processing time and computational requirements needed to solve some location problems (as the one in this chapter) may be considerable, the use of high performance computing techniques in location science is rather scarce, with hardly a dozen papers dealing with the topic. Usually, the proposed parallel approaches have been based on distributed programming paradigms executed on multicomputers [21, 29, 30, 39, 74, 134, 137, 138, 140, 144]. The

only exception seems to be [72], where an OpenMP implementation on a shared memory machine is addressed.

This section explores and evaluates a parallel implementation of UEGOf with application to the complex location problem described in Section 2.1. The parallel strategy has been developed to be executed on a multiprocessor system.

2.6.1 ParUEGOf

ParUEGOf presents a multithreaded approach of UEGOf. In multicore computers, all the processors have direct access to the whole memory. Processors are connected to an interconnection network, through which they can access the common memory banks. There exist several ways to deal with parallelism in a shared memory model (see Subsection 1.3.2). For the problem at hand, OpenMP [26] has been selected, since it is a portable and scalable model, and gives programmers a simple and flexible interface for developing parallel applications.

In ParUEGOf, the parallelism comes from the concurrent execution of two procedures: the creation and optimization methods. Notice that the computational load of UEGOf is concentrated in those procedures, and therefore, in order to exploit the architecture properly, they must be performed in parallel. It is important to highlight that there exists a "synchronization point" imposed by a 'selection' procedure. In such a procedure, only one thread will work with the whole species-list to maintain coherence in the data. Even so, to reduce the waiting time, partial selections are carried out concurrently, although in the end a global one performed in the 'selection' procedure is required for a correct performance of the algorithm in terms of quality in the solutions.

Algorithm 5 describes the basic structure of ParUEGOf. As can be seen, there are three parallel procedures. Each of them creates a parallel region with $th_{\max}$ threads. This value refers to the maximum number of available processors to solve the problem.

Initially, in *Init_species-list_parallel*, $th_{\max}$ species are randomly created, one at each processor, and they are optimized (as described below). This will allow to explore the whole area faster and fairly accurately.

Then, at each iteration of the algorithm, the creation procedure (*Create_species_parallel*) occurs in parallel distributing the current species-list among $th_{\max}$ threads. Notice that the species-list is shared by all threads which read from and write in it

Algorithm 5: ParUEGOf

```
1: Init_species-list_parallel
2: for t = 1 to t_max do
3:     Create_species_parallel
4:     Selection
5:     Optimize_species_parallel
6:     Selection
```

inside the parallel region. Threads only receive the memory address of the corresponding individuals and they are in charge of either reading or updating through this value.

The procedure for accessing the current species-list is as follows. Each thread picks up one specie from the current species-list and applies the corresponding genetic operators to it, obtaining a new offspring. Once a particular thread finishes its task creating an offspring, it picks up another individual from the global species-list, until all the species have been picked up by some thread. It is important to mention that some threads may be faster than others, so they may work with more species. This method to balance the global species-list between threads is known as dynamic schedule with chunk one in OpenMP literature. As a result, each thread has its own new candidate sub-lists, which must be included in the global species-list. But before that, since there may be many new candidate solutions, in order to reduce the computational load of the global selection procedure (Step 4), a partial selection is applied by each thread to its own new candidate sub-list. After that procedure, the selected species are included in the global species-list. Notice that this is a critical region and it is necessary to ensure that the threads do not update the global species-list simultaneously.

In the parallel optimization procedure (*Optimize_ species_ parallel*), each individual of the current species-list is optimized using the Weiszfeld-like local optimizer described in Section 2.5. The distribution of species and the load balancing technique are similar to the ones described above.

As the results show, for the problem at hand, the use of this paradigm allows to obtain good efficiencies.

2.6.2 Computational studies

In order to evaluate the performance of ParUEGOf, a representative set of location problems has been considered, varying the number $i_{\max}$ of demand points, the number $j_{\max}$ of existing facilities and the number k of those facilities belonging to the chain. In particular, both large ($500 \leq i_{\max} \leq 1000$) size problems described in Table 2.1 (Page 46) as well as larger size problems ($i_{\max} \geq 5000$) described in Table 2.9 have been considered. All the computational studies presented in this section have been obtained in the shared memory machine Ben of the Supercomputer Center of Murcia, Spain (see Subsection 1.3.5).

Study of the reliability of ParUEGOf

The aim of this subsection is to show that ParUEGOf works the same as UEGOf, in the sense that they both obtain similar solutions. Since both algorithms are heuristics, they may provide different solutions, so they have been run five times for each problem. The average time (in seconds) in the five runs, the average objective function value (columns Π) and the maximum Euclidean distance between any pair of solutions in the five runs (columns $MaxDist$) have been computed.

Table 2.12 summarizes the results for UEGOf and ParUEGOf with different number of processors, ParUEGOf(P). For the sake of completeness, values for UEGO have also been included. The given results correspond to the mean values for all the problems grouped for each value of $i_{\max}$. As can be seen, the differences between any pair of solutions in the five runs are always negligible for both UEGOf and ParUEGOf (and UEGO), which shows their robustness (see columns $MaxDist$). And of course, the solution obtained by all the algorithms is practically the same (see columns Π). The slight differences in objective function value are due to the high steepness of the objective function value around the optimal solution, as commented before. As was mentioned in Section 2.5, UEGO has not been able to solve the largest problems, i.e. those with setting $i_{\max} \geq 5000$.

Study of the efficiency of ParUEGOf

This subsection shows that the simple parallel algorithm ParUEGOf can reduce the CPU time of UEGOf proportionally to the number of available processors while obtain-

Algorithm	Time	Π	$MaxDist$	Time	Π	$MaxDist$
	$i_{\max} = 500$			$i_{\max} = 1000$		
UEGO	428	21.903	0.000	1015	6.684	0.000
UEGOf	286	21.903	0.000	352	6.683	0.000
ParUEGOf(2)	146	21.902	0.000	178	6.684	0.000
ParUEGOf(4)	76	21.903	0.000	90	6.683	0.000
ParUEGOf(8)	40	21.903	0.000	48	6.683	0.000
ParUEGOf(16)	23	21.902	0.000	27	6.684	0.000
ParUEGOf(32)	14	21.902	0.000	17	6.684	0.000
ParUEGOf(64)	10	21.902	0.000	12	6.684	0.000
ParUEGOf(128)	9	21.902	0.000	11	6.684	0.000
	$i_{\max} = 5000$			$i_{\max} = 10000$		
UEGO	-	-	-	-	-	-
UEGOf	1727	-31.663	0.000	2914	-21.299	0.000
ParUEGOf(2)	869	-31.663	0.000	1463	-21.299	0.000
ParUEGOf(4)	436	-31.663	0.000	736	-21.299	0.000
ParUEGOf(8)	226	-31.663	0.000	378	-21.298	0.000
ParUEGOf(16)	123	-31.663	0.000	199	-21.299	0.000
ParUEGOf(32)	72	-31.664	0.000	111	-21.298	0.000
ParUEGOf(64)	48	-31.663	0.000	70	-21.299	0.000
ParUEGOf(128)	37	-31.663	0.000	52	-21.299	0.000

Table 2.12: Average results for large problems.

ing the same solutions as UEGOf. In particular, the efficiency measure of the parallel algorithm, which estimates how well-utilized the processors are in solving the problem, is computed.

As can be seen in Table 2.13, the efficiency of ParUEGOf shows an almost ideal efficiency, with values close to 1 for up to $P = 8$ processors. Additionally, the efficiency values decrease when the number of processors increases. This tendency, may be explained by the parallel overheads and the non-parallelizable (sequential) parts, which mean idle processors during the optimization procedure. These two adverse effects increase as the number of processors does. Furthermore, the scalability of ParUEGOf is good, since its efficiency improves as the difficulty of the problem increases (with the number $i_{\max}$ of demand point).

Noteworthy is that nowadays most PCs have a multicore architecture with up to 8 processors, and hence it is possible to use all their computing power to solve the

	$i_{\max}$			
P	500	1000	5000	10000
2	0.98	0.99	1.00	1.00
4	0.94	0.98	0.99	0.99
8	0.89	0.93	0.96	0.96
16	0.77	0.82	0.88	0.92
32	0.62	0.65	0.75	0.82
64	0.43	0.44	0.56	0.65
128	0.25	0.26	0.36	0.44

Table 2.13: Average efficiency for large problems.

location problem at hand. The current standalone multicore personal computers have been successfully used in other fields to accelerate sequential codes [1, 2], but they still have not been fully exploited in the location field. As far as we know, only in [32] are the four cores of an Intel Core 2 Quad CPU put to work in parallel to solve an uncapacitated warehouse location problem, although no special strategy is used there: the same heuristic algorithm is run in the four cores, as a kind of multi-start strategy. For the sake of completeness, ParUEGOf has also been executed on a PC with 8 cores Intel Xeon at 1.87 GHz with 16 GB of RAM shared memory. The conclusions were similar than the ones previously inferred. More precisely, the CPU time required for solving a given instance can be divided by 8 as compared to the case when only one processor is used (as with the sequential algorithm). In that case, the whole architecture would be exploited by ParaUEGOf properly, and at the same time, the solution obtained is as reliable as the sequential version. For more information about this study, the interested reader is referred to [7].

2.7 Conclusions

Location Science is an important topic of research. The selection of the right location and quality for a new facility is crucial to its success, especially when the facility has to *compete* for the customers' demand with other facilities. To find the optimal site, the mathematical model describing the characteristics of the location problem at hand should be as realistic as possible. In this chapter it has been researched to what

extent the assumption of fixed demand, commonly employed in competitive location literature, has an impact on the location decision. To our knowledge, existing literature says nothing about this impact. It has been found that this assumption greatly affects the optimal solution and the optimal value of the problem. Although this assumption helps to make the problems more computationally tractable, it should only be used when the demand is really fixed. Otherwise, the solutions obtained may not be reliable at all.

On the other hand, the use of global optimization techniques such as those presented in this chapter, may be of help when solving the problems. Both the interval B&B method and the evolutionary algorithm UEGO can handle difficult location problems. The first one is an exact method that can solve nearly any continuous optimization problem, although it can only solve small instances. The second one can generate solutions of large instances. To be effective, it requires a suitable local optimizer for the problem at hand. Of course, depending on the problem at hand, other optimization techniques may also be useful.

A modification of the local search has also been proposed. Such a modification allows, on the one hand, to save between 1% to 37% of the CPU time when solving small problems and around 50% in large size problems, and on the other hand, to tackle larger problems than the original version UEGO.

Additionally, a parallel implementation of the new heuristic UEGOf has been developed, ParUEGOf; this parallel algorithm reduces the CPU time proportionally to the number of processing elements (at least up to 8 processors). The new method and its parallelization are robust, in the sense that they always find the same solution in all the runs.

CHAPTER 3

A planar location and design leader-follower problem with endogenous demand

The scenario considered in this chapter is that of a *duopoly*. A chain (the leader) wants to set up a new facility in the market, where similar facilities of a competitor (the follower), and possibly of its own chain, are already present. The demand is supposed to be endogenous and concentrated at some known demand points. Both the location and the quality of the new facility are to be found. The follower will react by locating another facility after the leader locates its own facility. The objective of the leader is to find the location and the quality of its new facility that maximizes its profit, following the location of the facility of the follower. This model is the extension of that of the previous chapter to the case in which the competitor's chain reacts by also locating a new facility, and it is a (1|1) centroid problem according to Hakimi's nomenclature (see Section 1.1).

As in the previous chapter, the feasible set is considered to be a region of the plane, the patronizing behavior of customers is assumed to be probabilistic, and the quality of the facility is regarded as a third decision variable of the model. And for the first time in the literature on centroid problems, endogenous demand is contemplated. This problem is a hard-to-solve global optimization problem, with many local maxima and in some instances with very different objective values at quite close feasible points, as will be shown.

The literature on centroid problems is scarce (see [56] for a review on the topic until 1996), and to our knowledge, among the existing papers only five of them deal with continuous problems. This is mostly due to the complexity of that type of bi-level programming problems. Drezner [44] solved the (1|1) centroid problem for the Hotelling model and Euclidean distances exactly, through a geometric-based approach. Bhadury *et al.* [14] also considered the $(nff|nfc)$ centroid problem for the Hotelling model with

Euclidean distances, and gave an alternating heuristic to cope with it. Drezner and Drezner [45] considered the Huff model, and proposed three heuristic approaches for handling the (1|1) centroid problem (see also [46]). More recently, Redondo *et al.* [133] introduced four heuristics for handling a (1|1) centroid problem with Huff patronizing behavior and with the quality of the new facility as a variable of the problem. In all those papers the demand was assumed to be exogenous.

The (1|1) centroid problem considered in this thesis is similar to that in [133], but in which the demand varies depending on the attraction for the facilities.

It is important to mention that to solve a single centroid problem, many medianoid problems have to be solved, since the evaluation of the leader's objective function at a given point requires the resolution of the corresponding medianoid problem. Of course, it is important to compute the leader's objective function value accurately, which means that the follower's problem has to be solved with precision. However, the medianoid problem (which is the problem introduced in the previous chapter) is also a hard-to-solve global optimization problem. In the previous chapter, both an iB&B method and the evolutionary algorithm UEGO were adapted to solve the medianoid problem, and both methods will be considered here as alternatives to deal with the medianoid problems.

The chapter is organized as follows. In Section 3.1, the centroid problem is introduced. Section 3.2 describes three procedures for solving it, namely, a grid search procedure, a multistart heuristic and an evolutionary algorithm called TLUEGO. Additionally, a local search procedure, called SASS+WLMv, is proposed to be used in both the multistart and the evolutionary algorithms. In order to compare the performance of the proposed algorithms, a comprehensive computational study is carried out. As will be seen, the results show that the evolutionary algorithm TLUEGO is the best choice to solve the centroid problem. In Section 3.3, special attention is paid to the *fuse* procedure of TLUEGO and an extensive computational study devoted to studying the influence of that *fuse* procedure in the quality of the solutions is performed. In order to reduce the CPU time needed by TLUEGO for solving the centroid problem with higher quality, three parallelizations of TLUEGO are presented in Section 3.4. Finally, the main findings in the chapter are highlighted in Section 3.5.

3.1 The model

A chain, the *leader*, wants to locate a new single facility in a given area of the plane, where $j_{\max}$ facilities offering the same goods or product already exist. The first k (≥ 0) of those $j_{\max}$ facilities belong to the chain, and the other $j_{\max} - k$ (> 0) to a competitor chain, the *follower*. The leader knows that the follower, as a reaction, will subsequently position a new facility too. The demand, supposed to be endogenous, is concentrated at $i_{\max}$ demand points, whose locations $locd_i$ are known. The location $locf_j$ and quality of the existing facilities are also known.

The following notation will be used throughout this chapter:

Indices

- i index of demand points, $i = 1, \ldots, i_{\max}$.
- j index of existing facilities, $j = 1, \ldots, j_{\max}$.

Variables

- $z_1 = (x_1, y_1)$ location of the new leader's facility.
- α_1 quality of the new leader's facility.
- $nf_1 = (z_1, \alpha_1)$ variables of the new leader's facility.
- $z_2 = (x_2, y_2)$ location of the new follower's facility.
- α_2 quality of the new follower's facility.
- $nf_2 = (z_2, \alpha_2)$ variables of the new follower's facility.

Data

- $locd_i$ location of the i-th demand point.
- $locf_j$ location of the j-th existing facility.
- $d_i^{\min}$ minimum distance from $locd_i$ at which the new facilities can be located.
- $d_{i,j}$ distance between demand point i and facility j.
- $a_{i,j}$ quality of facility j as perceived by demand point i.
- $g_i(\cdot)$ a non-negative non-decreasing function.
- $u_{i,j}$ attraction that demand point i feels for facility j (or utility of j perceived by the people at i), $u_{i,j} = \dfrac{a_{i,j}}{g_i(d_{i,j})}$
- γ_i weight for the quality of the new facilities as perceived by demand point i.
- FR_1 location space where the leader will locate its new facility.

$\alpha_1^{\min}$	minimum level of quality for the new leader's facility.
$\alpha_1^{\max}$	maximum level of quality for the new leader's facility.
FR_2	location space where the follower will locate its new facility.
$\alpha_2^{\min}$	minimum level of quality for the new follower's facility.
$\alpha_2^{\max}$	maximum level of quality for the new follower's facility.

Miscellaneous

$d_i(z_l)$	distance between demand point i and $z_l, l = 1, 2$.
u_{i,nf_l}	attraction that demand point i feels for $nf_l, l = 1, 2$, $u_{i,nf_l} = \dfrac{\gamma_i \alpha_l}{g_i(d_i(z_l))}$.
U_i	total utility perceived by a customer at demand point i.
$w_i(U_i)$	demand (or buying power or total expenditure) at demand point i.
$M_1(nf_1, nf_2)$	market share obtained by the leader after the location of the new facilities.
$M_2(nf_1, nf_2)$	market share obtained by the follower after the location of the new facilities.
$\Pi_1(nf_1, nf_2)$	profit obtained by the leader after the location of the new facilities.
$\Pi_2(nf_1, nf_2)$	profit obtained by the follower after the location of the new facilities.

It is assumed that $g_i(d_{i,j}) > 0 \ \forall i, j$. As in the previous chapter, the demand is endogenous and it is assumed to be a function of $U_i = u_{i,nf_1} + u_{i,nf_2} + \sum_{j=1}^{j_{\max}} u_{i,j}$, $w_i(U_i) = w_i^{\min} + incr_i \cdot e_i(U_i)$, where $incr_i = w_i^{\max} - w_i^{\min}$, and $w_i^{\max}$ (resp. $w_i^{\min}$) denotes the maximum (resp. minimum) possible demand at $locd_i$. Again, function $e_i(U_i)$ can be interpreted as the share of the maximum possible increment that a customer decides to expend under a given location scenario.

Using these assumptions, the market share attracted by the leader's chain after the location of the leader and the follower's new facilities is

$$M_1(nf_1, nf_2) = \sum_{i=1}^{i_{\max}} w_i(U_i) \frac{u_{i,nf_1} + \sum_{j=1}^{k} u_{i,j}}{u_{i,nf_1} + u_{i,nf_2} + \sum_{j=1}^{j_{\max}} u_{i,j}},$$

and the corresponding market share attracted by the follower's chain is

$$M_2(nf_1, nf_2) = \sum_{i=1}^{i_{\max}} w_i(U_i) \frac{u_{i,nf_2} + \sum_{j=k+1}^{j_{\max}} u_{i,j}}{u_{i,nf_1} + u_{i,nf_2} + \sum_{j=1}^{j_{\max}} u_{i,j}}.$$

Given nf_1, the problem for the follower is the $(1|nf_1)$ medianoid problem:

$$(FP(nf_1)) \quad \begin{cases} \max & \Pi_2(nf_1, nf_2) = F_2(M_2(nf_1, nf_2)) - G_2(nf_2) \\ \text{s.t.} & z_2 \in FR_2 \\ & d_i(z_2) \geq d_i^{\min}, i = 1, \ldots, i_{\max} \\ & \alpha_2 \in [\alpha_2^{\min}, \alpha_2^{\max}] \end{cases}$$

whose objective is the maximization of the profit obtained by the follower (once the leader has set up its new facility at nf_1), to be understood as the difference between the revenues obtained from the captured market share minus the operating cost of the new facility (see [61]). F_2 is a strictly increasing function which transforms the market share into expected sales and G_2 is a function which gives the operating cost for the follower of a facility located at z_2 with quality α_2.

Let $nf_2^*(nf_1)$ denote an optimal solution for $(FP(nf_1))$. The problem for the leader is the $(1|1)$ centroid problem:

$$(LP) \begin{cases} \max & \Pi_1(nf_1, nf_2^*(nf_1)) = F_1(M_1(nf_1, nf_2^*(nf_1))) - G_1(nf_1) \\ \text{s.t.} & z_1 \in FR_1 \\ & d_i(z_1) \geq d_i^{\min}, i = 1, \ldots, i_{\max} \\ & \alpha_1 \in [\alpha_1^{\min}, \alpha_1^{\max}] \end{cases} \tag{3.1}$$

where F_1 and G_1 are the corresponding expected sales and operating costs functions, respectively, for the leader's chain.

In the computational studies presented in this chapter, the following choices were made:

- Functions $F_l,\ l = 1, 2$, are linear, $F_l(M_l) = s_l \cdot M_l$, where s_l is the income per unit of goods sold.

- Usually, the operating costs of a new facility consist of the sum of the locational costs and the costs related to reaching a given level of quality. Therefore functions $G_l, l = 1, 2$, are assumed to be separable, in the form $G_l(nf_l) = G_l^a(z_l) + G_l^b(\alpha_l)$.

In particular, it has been considered $G_l^a(z_l) = \sum_{i=1}^{i_{\max}} \Phi_l^i(d_i(z_l))$, with $\Phi_l^i(d_i(z_l)) = Aver_{A_i}(w_i(U_i))/((d_i(z_l))^{\phi_l^{i0}} + \phi_l^{i1})$, $\phi_l^{i0}, \phi_l^{i1} > 0$ and $G_l^b(\alpha_l) = e^{(\alpha/\alpha_l^0 + \alpha_l^1)} - e^{\alpha_l^1}$, with $\alpha_l^0 > 0$ and $\alpha_l^1 \in \mathbb{R}$ as given values. Notice that in the cost function $G_l^a(z_l)$, $Aver_{A_i}(w_i(U_i))$ stands for the average value of $w_i(U_i)$ over the feasible set (the mathematical formulation is given in Section 2.2, see Equation (2.2)) and can be thought of as an estimation of the demand at $locd_i$ by a fixed number.

- *Linear expenditures* is considered (see Section 2.1), i.e., $w_i^{\min} = 0$, $w_i(U_i) = w_i^{\max} \cdot e_{i_1}(U_i)$, where $e_{i_1}(U_i) = c_i U_i$, with c_i a given constant such that $c_i \leq 1/U_i^{\max}$, where $U_i^{\max}$ is the maximum utility that can possibly be perceived by a customer at i.

A more detailed explanation of the parameters and functions of the model, as well as other possible expressions for F_l and G_l, can be found in [61]. Of course, other functions might be more suitable depending on the real problem considered, and for each real application the most appropriate F_l and G_l functions should be discovered. In [154] the interested reader can find a pseudo-real application to the case of the location of supermarkets in the Autonomous Region of Murcia, in Southern Spain. Although in that paper the demand was assumed to be exogenous and no reaction from the competitor was expected, the parameters and functions have the same meaning as those in this chapter.

As can be seen, the leader problem (LP) is much more difficult to solve than the follower problem $(FP(nf_1))$. Notice, for instance, that to evaluate its objective function Π_1 at a given point nf_1, first the corresponding medianoid problem $(FP(nf_1))$ to obtain $nf_2^*(nf_1)$ must be solved. Furthermore, in order to compute the objective value of Π_1 at nf_1 accurately, the follower problem $(FP(nf_1))$ has to be precisely solved since otherwise, the error of the approximate value can be considerable.

As shown in Chapter 2, the objective function of the follower's problem with exogenous demand is multimodal, but it tends to be smoother than the objective function of the follower's problem with endogenous demand, which has many more local optima and whose landscape is much steeper. Of course, the complexity of the centroid problem is more greatly affected due to the endogenous demand assumption.

In order to show the difficulty of the problem at hand, and its differences with the exogenous demand case, the quasi-real example introduced in [154] dealing with

the location of supermarkets in an area around the city of Murcia has been solved. In particular, a working radius of 25 Km around the city of Murcia was considered. In all, 632,558 people live in that area, and they form our set of customers. Although they are distributed over 71 population centers, with populations varying between 1,138 and 178,013, in this example it has been considered an aggregated version, in which only population centers with a city hall are taken into account. The 21 towns with a city hall form our reduced set of demand points, with the population obtained by aggregating all population centers in the town which they administratively depend on. The mean purchasing power of a town was considered proportional to its population. The position and population of the towns can be seen in figures 3.1 and 3.2, where yellow circles represent the forbidden areas around the existing demand points, which are at the center of those circles (the greater the circle, the greater the purchasing power at the demand point). The location space $FR_1 = FR_2$ was taken as the smallest rectangle containing all demand points.

There are five supermarkets in the area: three from a first chain, 'E', and two from another chain, 'C'. Those figures show the location of each supermarket as viewed from chain E's point of view: firms belonging to chain E are marked by a black triangle (▲), and firms from the other chain are shown by a black square (■) on the map. The quality parameters $a_{i,j}$ have been set within the interval [3,4]. The optimization of quality for the new facilities was carried out in the interval $[\alpha_l^{\min}, \alpha_l^{\max}] = [0.5, 5], l = 1, 2$. The income per unit of goods sold has been set to $s_l = 32,\ l = 1, 2$. Due to the lack of real data from the chains (they consider that data sensitive for them and are not willing to make them public), the other parameters have been validated in an ad hoc way to obtain 'reasonable' results. The interested reader is referred to [154] for more details about the case study and the value of the parameters.

The same problem was solved both assuming fixed demand and considering endogenous demand (as in Section 2.2 for the follower problem). In Figure 3.1, the optimal location and quality for the new leader's facility (represented by $*$) and the new follower's facility (represented by $+$), when chain E is the leader, assuming that the demand is exogenous (as obtained by algorithm UEGO_cent.SASS [133]) can be seen. The corresponding solutions when the demand is endogenous are shown in Figure 3.2 (as obtained using TLUEGO_UE, see Subsection 3.2.3). In those figures, two windows on the right and bottom of the map have been added, allowing the reader to view all

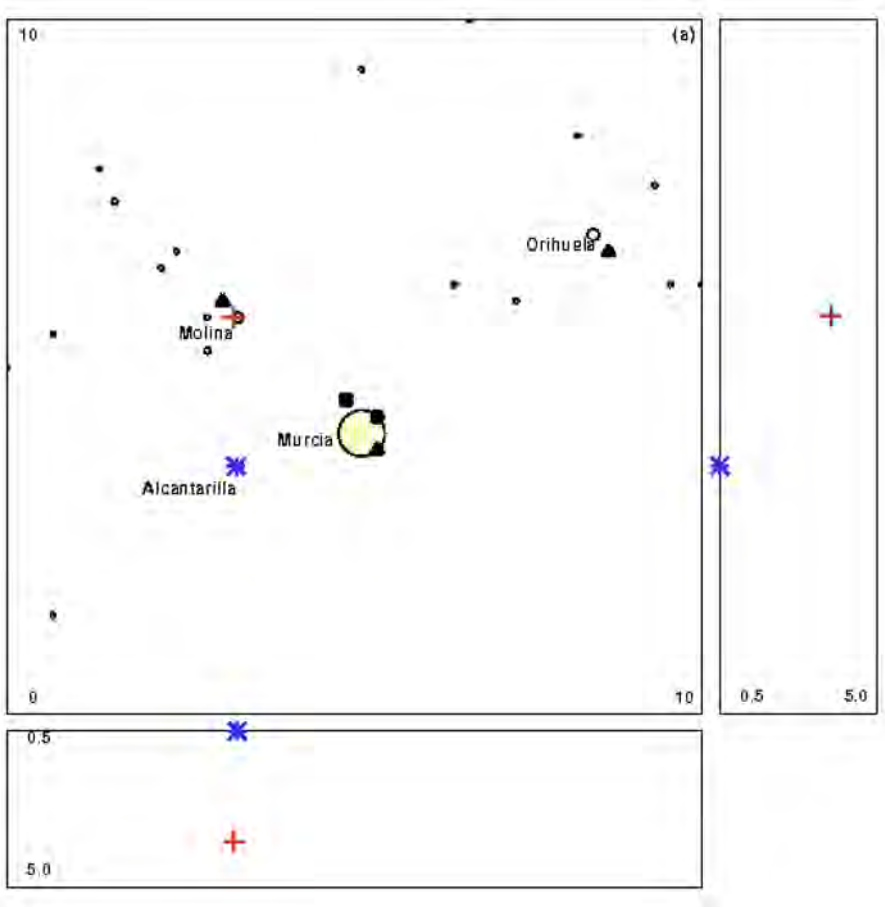

Figure 3.1: Optimal location and quality for both leader and follower when chain E is the leader. Exogenous demand. Leader's facility $\ast$ (blue) and follower's facility $+$ (red).

three 2-dimensional projections of the 3-dimensional solution set: the map itself shows the 2-dimensional spatial part, without the quality, the right pane shows the quality and vertical space part (quality increases from left (0.5) to right (5)), the bottom pane shows the quality and horizontal space part (quality increases from top (0.5) to bottom (5)). The numerical results are shown in Table 3.1.

As can be seen, in the exogenous demand case, the optimal location for the leader is near the city of Alcantarilla, with a quality of 0.5. At that point, the market share captured by the new leader's facility is $m_1 = 2.112$, which is 5.94% of the total market share. Considering all its facilities, chain E gets 53.22% of the market, and a profit $\Pi_1 = 593.352$. The location for the follower's facility is near the city of Molina, with a quality of 3.696, where it captures 20.04% of the total market share. However, the

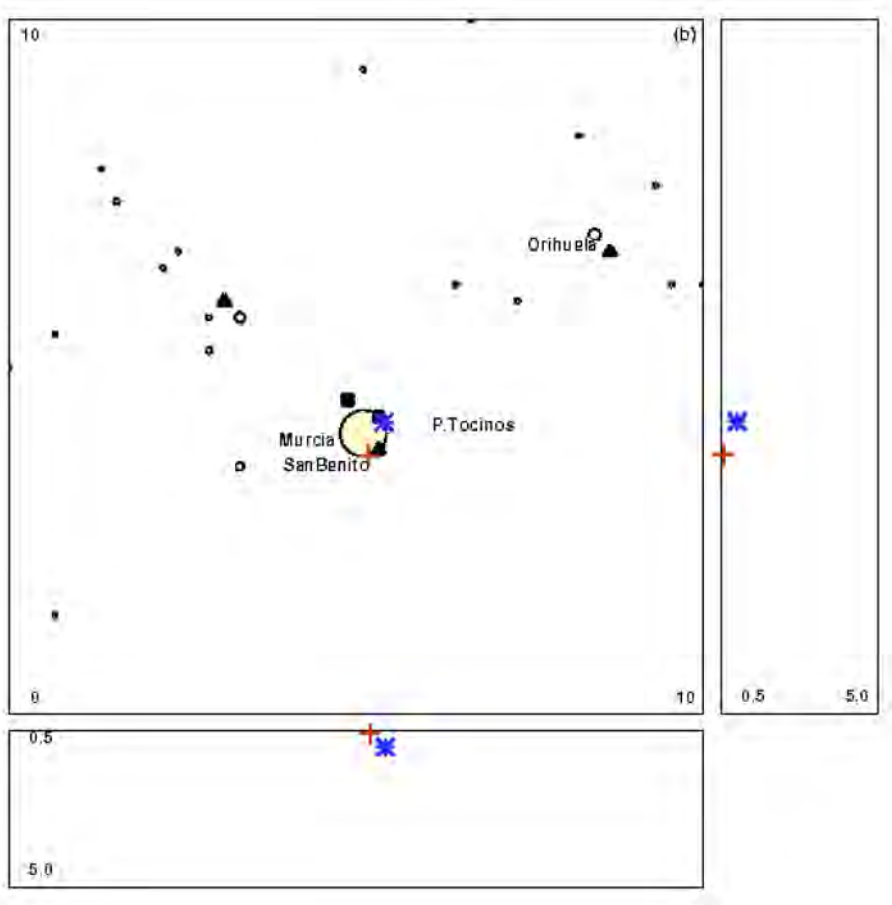

Figure 3.2: Optimal location and quality for both leader and follower when chain E is the leader. Endogenous demand. Leader's facility $\ast$ (blue) and follower's facility $+$ (red).

Demand	nf_1	M_1	m_1	Π_1	nf_2	M_2	m_2	Π_2
				Leader: chain E				
Exogenous	(3.303, 6.433, 0.500)	18.915	2.112	593.352	(3.259, 4.285, 3.696)	16.625	7.123	461.776
Endogenous	(5.407, 5.798, 0.961)	2.807	0.419	73.454	(5.190, 6.276, 0.571)	3.618	0.249	101.563
				Leader: chain C				
Exogenous	(8.487, 3.026, 3.277)	15.961	6.247	442.122	(3.274, 6.441, 0.500)	19.579	2.187	614.652
Endogenous	(5.368, 6.166, 1.042)	3.822	0.453	106.320	(5.298, 6.228, 0.571)	2.6378	0.2489	70.227

Table 3.1: Examples.

leader's optimal location in the variable demand case is in the suburb of Puente Tocinos, in Murcia city, with a quality of 0.961. The market share captured by the facility is 0.419, which is 5.94% of the total market share. The whole chain gets 43.68% of the market and a profit $\Pi_1 = 73.454$. The location for the follower's facility is near the

suburb of San Benito, in Murcia city, with a quality of 0.571, where it captures 3.875% of the total market share.

If it is now assumed that chain C is the leader, then, in the exogenous demand case, the optimal location for the leader is near the city of Orihuela, with a quality of 3.277, where the facility captures 17.57% of the total market share. The location for the follower's facility is near the city of Alcantarilla, with a quality of 0.5, where it captures 6.15% of the total market share. The corresponding total market share captured by the chains and their profits can be seen in Table 3.1. However, the leader's optimal location in the endogenous demand case is near the suburb of San Benito, in Murcia city, with a quality of 1.042, and the location for the follower's facility is near the suburb of San Benito too, with a quality of 0.571.

These two examples show how important it is to take endogenous demand into consideration. As can be seen, the maximum profit for the chain is obtained at different locations and with different qualities, depending on whether endogenous demand or exogenous demand is considered. Also, the percentage of market share captured by the chains may change to the point that the chain obtaining more profit may be the opposite one.

3.2 Solving the centroid problem

In this section, three heuristics devised to cope with the centroid problem are described. More precisely, a grid search procedure, a multistart method named MSH, and an evolutionary algorithm called TLUEGO, are presented. In the last two, a local optimizer is needed. A subsection will be dedicated to briefly explain such a local technique. Two variants have been designed for the local optimizer, which derive two versions for MSH and TLUEGO algorithms.

3.2.1 GS: a grid search procedure

The first method is a simple Grid Search procedure (GS) as in [133]. A grid of points that cover the leader's 3-dimensional searching region is generated. For each point of the grid its feasibility is checked. If it is feasible, then the objective function for the leader is evaluated, which implies knowing the optimal solution for the follower and

therefore the corresponding medianoid problem must be previously solved. To this aim, the algorithm UEGO described in Subsection 2.3.1 is used. When all the feasible points of the grid have been evaluated, a second finer grid is constructed in the vicinity of the point of the first grid having the best objective value. In this work, the length of the step between two adjacent points in the first grid was 0.1 units in each coordinate, and 0.02 unit in the second grid.

3.2.2 The local optimizer SASS+WLMv

As mentioned above, to solve the centroid problem, both the multistart and the evolutionary algorithms make use of a local procedure. Local optimizers usually assume that the configuration of the problem during the optimization process does not change. However, this is not the case for the centroid problem, since every time the leader's facility changes so does the follower's facility. Thus, the value of the objective function of the leader's problem may change if the new configuration is taken into account. This means that the new follower's facility should be computed every time the leader's facility changes. However, considering that the number of function evaluations in any local optimizer is usually large, obtaining the exact new follower's facility at each new location of the leader's facility will make the process very time-consuming.

To deal with the centroid problem with exogenous demand, a local procedure called SASS+WLM was introduced in [133]. The idea of such an algorithm is to apply the stochastic hill climber SASS (see [147]) to improve the leader's facility, and a Weiszfeld-like algorithm WLM to approximate the follower's. The leader optimization is focused on a sphere whose radius is determined by the input parameter σ_{ub}. The algorithm stops when a maximum number of iterations ($iter_{\max}$) is reached, or when a maximum number of consecutive failures at improving the objective function ($Maxfcnt$) occurs.

For the problem at hand, and after trying different strategies, a local procedure similar to SASS+WLM in [133] is proposed. The pseudocode of this new method is given in Algorithm 6. The main differences between the local algorithm used in this thesis (which will be called SASS+WLMv) and the one in [133] are:

- The Weiszfeld-like algorithm used now for updating the follower's facility is WLMv (described in Subsection 2.3.2), instead of WLM. Similar to what was considered for UEGO (see Subsection 2.3.1), WLMv stops when either two con-

Algorithm 6: Algorithm SASS+WLMv($nf_1, nf_2, iter_{\max}(=15), \sigma_{ub}$)

1: Initialize SASS parameters. Set $iter = 1, nf_1^{opt} = nf_1, \Pi_1^{opt} = \Pi_1(nf_1, nf_2)$.
2: **while** $iter \leq iter_{\max}$ **do**
3: Update SASS parameters considering the previous successes at improving the objective function value of the leader.
4: Generate a location for the leader $nf_1^{(iter)}$ within the updated radius.
5: Solve the corresponding medianoid problem using WLMv and let $nf_2^{(iter)}$ denote the solution obtained.
6: **if** $\Pi_1(nf_1^{(iter)}, nf_2^{(iter)}) > \Pi_1^{opt}$ **then**
7: set $nf_1^{opt} = nf_1^{(iter)}$ and $\Pi_1^{opt} = \Pi_1(nf_1^{(iter)}, nf_2^{(iter)})$.
8: $iter = iter + 1$.
9: Compute the corresponding follower nf_2^{opt} for nf_1^{opt} using either iB&B or UEGO.
10: **if** $\Pi_1(nf_1^{opt}, nf_2^{opt}) > \Pi_1(nf_1, nf_2)$ **then**
11: return (nf_1^{opt}, nf_2^{opt})
12: **else**
13: Return (nf_1, nf_2).

secutive iterations are closer than the tolerance $\epsilon_1 = \epsilon_2 = 0.0001$, or when a maximum number of $ic_{max} = 400$ iterations is reached.

- The WLMv algorithm is not as reliable as the corresponding method WLM for the fixed demand case. Then, a large maximum number of iterations $iter_{\max}$ in SASS could direct the leader towards overestimated solutions. To deal with this drawback, the parameter $iter_{\max}$ in SASS+WLMv is reduced to 15. Additionally, once the maximum number of iterations $iter_{\max}$ is reached, the medianoid problem is solved accurately. Otherwise the objective value for the leader could be completely wrong. This can even happen if the solutions are very close to optimality in objective function value but are in significantly different locations, and even if the leader's problem is solved optimally given the non-optimal follower's solution. For the problem at hand, two alternatives have been considered when solving the medianoid problem: iB&B [61] or UEGO (see Section 2.3), resulting in two versions of the local optimizer.

Notice that the algorithm iB&B gives a list of small 3-dimensional intervals where any optimizer point must lie as a solution. Then, when selecting this method, in Step

9 of Algorithm 6, the solution nf_2^{opt} considered will be the best point evaluated by the algorithm iB&B.

3.2.3 TLUEGO: A two-level evolutionary global optimization algorithm

The evolutionary algorithm TLUEGO is similar to the algorithm UEGO_cent.SASS introduced in [133], which deals with the corresponding centroid problem with exogenous demand. TLUEGO, as well as UEGO_cent.SASS, shares some concepts and ideas with UEGO (see Subsection 2.3.1). In particular, the concept of species (including attraction radius) and the use of a local optimizer have been either adopted or adapted to cope with the centroid problems. The values of the input parameters used for UEGO (see Subsection 2.3.3) have also been adopted for TLUEGO.

In the following, the general structure of TLUEGO is provided (see Algorithm 7). At the beginning, a single species (the root) exists, and as the algorithm evolves and applies genetic operators, new species can be created. For TLUEGO to work properly, it is very important to correctly evaluate the fitness of the new species after the *creation* procedure. To this aim, a reliable follower solution has to be computed, and to do so, two alternative algorithms are possible: iB&B or UEGO. At every generation, TLUEGO performs a *local optimizer* operation on each species. For the problem at hand, the algorithm SASS+WLMv is used. Notice that it is executed twice in order to have more chances of obtaining a better point. The value of σ_{ub} passed to SASS+WLMv is always (the two times it is called) the radius associated to the calling species. In this way, the scope of the local optimizer is exactly the area covered by the species. Notice that a species involves a 'cooling' technique which enables the search to focus on the promising regions of the space, starting off with a relatively large radius that decreases as the search proceeds. Therefore, exploration and exploitation of the search space is guaranteed. TLUEGO has been executed with the two variants of the local optimizer, i.e. considering iB&B and UEGO when computing a reliable solution for the follower (Step 9 in Algorithm 6). It is important to highlight that TLUEGO performs two *selection* procedures during the optimization process. The first one is carried out after the new offspring is generated. It consists of the 'Fuse species' and the 'Shorten species list' procedures. The second one takes place after the optimization procedure, and only

Algorithm 7: Algorithm TLUEGO($FE_{\max}$, $L_{\max}$, $t_{\max}$, $R_{t_{\max}}$)

1: Set iteration counter $t = 1$.
2: Initialize a random leader location (center of initial species) $nf_1^{(t)}$ and compute the corresponding follower $nf_2^{(t)}$ using either iB&B or UEGO.
3: $(nf_1^{(t)*}, nf_2^{(t)*})$ = SASS+WLMv$(nf_1^{(t)}, nf_2^{(t)}, iter_{\max}(= 15), \sigma_{ub}(= R_t))$.
4: (nf_1^{opt}, nf_2^{opt}) = SASS+WLMv$(nf_1^{(t)*}, nf_2^{(t)*}, iter_{\max}(= 15), \sigma_{ub}(= R_t))$.
5: **for** $t = 2$ until $t_{\max}$ **do**
6: Create new species.
7: Compute the corresponding follower for the new species using either iB&B or UEGO, and evaluate the leaders' fitness values.
8: Fuse species, and Shorten the species list.
9: **for** each existing species $nf_1^{(t)}$ (with radius R_t) and its corresponding follower $nf_2^{(t)}$ **do**
10: $(nf_1^{(t)*}, nf_2^{(t)*})$=SASS+WLMv$(nf_1^{(t)}, nf_2^{(t)}, iter_{\max}(= 15), \sigma_{ub}(= R_t))$.
11: (nf_1^{opt}, nf_2^{opt})=SASS+WLMv$(nf_1^{(t)*}, nf_2^{(t)*}, iter_{\max}(= 15), \sigma_{ub} = R_t)$.
12: Fuse species.
13: $t = t + 1$.
14: Return the best leader facility and its objective value.

considers the Fuse species procedure. In this work, in the Fuse procedure called at a level t, two species are fused into a single one whenever the distance between their centers is smaller than $2R_t$, where R_t is the radius associated at level t. The reader is referred to [133] for a more detailed description of these procedures.

The inclusion of iB&B or UEGO in TLUEGO derives two algorithms for solving the centroid problem, TLUEGO_BB and TLUEGO_UE, respectively.

3.2.4 MSH: A multistart heuristic algorithm

The MSH algorithm consists of randomly generating *MaxStartPoints* feasible candidate solutions for the leader and applying a local optimizer to them in order to achieve an optimized leader solution. The final solution will be the one with the best objective function value.

For the case at hand, the considered local optimizer has been SASS+WLMv (see Algorithm 6). Notice that this method focuses the search on an area defined by the parameter σ_{ub}. In order to provide a balance between exploitation and exploration of

the search space, this method has also been executed twice as in TLUEGO, but with different values for σ_{ub}. In the first call, a value of $\sigma_{ub} = 2.083895$ (the one corresponding to level 10 in TLUEGO) was considered. Such a value was chosen because in this way, the initial random candidate solutions in the multistart strategy can cover the whole searching space, and at the same time, they can focus on an area small enough so that the local procedure can find a good local optimum. In the second call, a value of $\sigma_{ub} = 0.162375$ (the one corresponding to level 23 in TLUEGO) was used to improve the quality of the local optimum obtained with the first call. These σ_{ub} values were selected after some preliminary studies, in which eight problems of different sizes were solved trying different strategies for the heuristic algorithm.

As in TLUEGO, two versions of the MSH method, called MSH_BB and MSH_UE, are proposed. They differ in whether iB&B or UEGO is used as a method of computing the follower nf_2^{opt} in Step 9 of the local optimizer SASS+WLMv (see Algorithm 6).

3.2.5 Computational studies

All the computational results in this section have been carried out on a processor Xeon IV with 2.4GHz and 1 GByte RAM. The algorithms have been implemented in C++.

To study the performance of the algorithms, 24 different problems have been generated varying the number $i_{\max}$ of demand points, the number $j_{\max}$ of existing facilities and the number k of those facilities belonging to the leader's chain. The actual settings $(i_{\max}, j_{\max}, k)$ employed are detailed in Table 3.2. For every setting, the problem was generated by randomly choosing its parameters uniformly within pre-specified intervals presented in Appendix. In all the problems, the following choices were made: $FR_1 = FR_2 = ([0, 10], [0, 10])$ and $\alpha_1, \alpha_2 \in [0.5, 5]$.

Since most of the proposed algorithms are heuristics, each run may provide a different solution. Thus, to study their robustness, for every heuristic algorithm, each

$i_{\max}$	15			25			50		
$j_{\max}$	2	5	10	2	5	10	2	5	10
k	0,1	0,1,2	0,2,4	0,1	0,1,2	0,2,4	0,1	0,1,2	0,2,4

Table 3.2: Settings of the test problems.

problem has been solved ten times and average values have been computed. Nevertheless, the heuristic GS has only been run once and the results obtained in that run (no average results) are given.

Tables 3.3, 3.4 and 3.5 show the results obtained by the algorithms when $i_{\max} = 15$, $i_{\max} = 25$ and $i_{\max} = 50$, respectively. In the column labelled '$Time$', the average time in the ten runs (in seconds) is given; in the 'Best Solution' column, the best solution (x_1, y_1, α_1) found in the ten runs is shown; the '$MaxDist$' column gives the maximum Euclidean distance (considering the three variables of the problem) between any pair of solutions provided by the algorithm, which gives an idea of how far the solutions provided by the algorithm in different runs can be; in the next three columns, the minimum, the average and the maximum objective value in the ten runs are given. Finally, in the 'Dev' column, the standard deviation is provided. As can be seen in these tables, two versions of TLUEGO and MSH algorithms have been executed. It is worth mentioning that the number of times that MSH_BB (resp. MSH_UE) was allowed to repeat its basic local optimizer was chosen so that the CPU time employed by MSH_BB (resp. MSH_UE) was, on average (when considering all the problems with the same value of $i_{\max}$), similar to the CPU time employed by TLUEGO_BB (resp. TLUEGO_UE) or a bit higher. In particular, for the problems with 15, 25 and 50 demands points, the number of starting points were 150, 200 and 250, respectively. On the bottom of these tables, the average results for each algorithm are showed.

The method considered to solve the medianoid problem does not seem to have too much influence on the quality of the final solution, i.e., TLUEGO and MSH behave similarly independent of whether iB&B or UEGO is employed. This fact is corroborated in Subsection 2.3.4, where it was stated that UEGO was able to obtain the global optimal solution for all the problems where iB&B could be executed (with $i_{\max} \leq 200$). However, the computing time is highly affected by those methods. The iB&B technique is faster than UEGO for small size problems, which helps to reduce the execution time of both TLUEGO and MSH. Namely, the use of iB&B reduces the computing time of TLUEGO for problems with $i_{\max} = 15$ (as compared to its counterpart executed with UEGO) by 74.6%. The corresponding reduction for the problems with $i_{\max} = 25$ is 30.9%. Similar reductions in computing time can be seen in MSH when iB&B is used instead of UEGO. Nevertheless, for medium size problems (with $i_{\max} = 50$ demand points), TLUEGO_UE and MSH_UE reduce the computing time as compared to

			Best Solution			Max	Objective Function			
($j_{\max}$,k)	Algorithm	$Time$	x_1	y_1	α_1	$Dist$	Min	Av	Max	Dev
(2,0)	TLUEGO_BB	618	8.505	4.154	0.50	0.049	-4.630	-4.630	-4.630	0.000
	TLUEGO_UE	1217	8.513	4.152	0.50	0.032	-4.630	-4.630	-4.630	0.000
	MSH_BB	619	8.558	4.138	0.50	0.258	-4.678	-4.653	-4.635	0.016
	MSH_UE	1254	8.566	4.133	0.50	0.235	-4.744	-4.663	-4.640	0.041
	GS	379932	8.540	4.140	0.50	-	-	-4.637	-	-
(2,1)	TLUEGO_BB	121	7.860	7.841	0.50	0.000	38.732	38.732	38.732	0.000
	TLUEGO_UE	977	7.860	7.841	0.50	0.001	38.731	38.732	38.732	0.001
	MSH_BB	142	7.865	7.841	0.50	0.111	38.607	38.667	38.727	0.043
	MSH_UE	1247	7.876	7.837	0.50	0.276	38.468	38.608	38.698	0.097
	GS	400226	7.860	7.840	0.50	-	-	38.730	-	-
(5,0)	TLUEGO_BB	264	5.731	8.062	0.88	0.058	-5.806	-5.805	-5.802	0.002
	TLUEGO_UE	1039	5.731	8.062	0.85	0.017	-5.804	-5.803	-5.802	0.001
	MSH_BB	314	5.773	8.078	0.81	0.525	-6.394	-6.115	-5.978	0.148
	MSH_UE	1188	5.729	8.064	0.91	3.315	-6.713	-6.095	-5.815	0.328
	GS	481907	5.720	8.080	0.75	-	-	-5.928	-	-
(5,1)	TLUEGO_BB	393	1.328	0.000	0.50	0.007	10.574	10.575	10.575	0.000
	TLUEGO_UE	1156	1.328	0.000	0.50	0.019	10.573	10.574	10.575	0.001
	MSH_BB	376	1.325	0.036	0.50	8.029	10.560	10.564	10.571	0.005
	MSH_UE	1159	1.328	0.004	0.50	0.036	10.571	10.573	10.574	0.001
	GS	511797	9.120	0.180	0.50	-	-	10.533	-	-
(5,2)	TLUEGO_BB	71	5.711	2.343	0.50	0.000	39.968	39.968	39.968	0.000
	TLUEGO_UE	691	5.714	2.341	0.50	0.003	39.968	39.969	39.970	0.001
	MSH_BB	109	5.711	2.345	0.50	0.162	39.852	39.910	39.965	0.042
	MSH_UE	1058	5.714	2.342	0.50	0.116	39.883	39.930	39.964	0.031
	GS	391193	5.720	2.340	0.50	-	-	39.952	-	-
(10,0)	TLUEGO_BB	140	0.000	1.854	0.50	0.000	-9.753	-9.753	-9.753	0.000
	TLUEGO_UE	726	0.000	1.854	0.50	0.001	-9.754	-9.753	-9.753	0.000
	MSH_BB	196	0.000	1.855	0.50	0.021	-9.758	-9.757	-9.756	0.001
	MSH_UE	984	0.000	1.854	0.50	0.017	-9.764	-9.758	-9.753	0.004
	GS	621003	0.120	1.880	0.50	-	-	-9.789	-	-
(10,2)	TLUEGO_BB	139	7.206	10.000	0.50	0.003	16.414	16.415	16.415	0.000
	TLUEGO_UE	709	7.206	10.000	0.50	0.000	16.415	16.415	16.415	0.000
	MSH_BB	198	7.206	9.995	0.50	0.080	16.392	16.405	16.411	0.007
	MSH_UE	915	7.206	10.000	0.50	0.009	16.402	16.411	16.415	0.005
	GS	573090	7.200	10.000	0.50	-	-	16.395	-	-
(10,4)	TLUEGO_BB	59	2.273	0.487	0.50	0.000	38.323	38.323	38.323	0.000
	TLUEGO_UE	613	2.273	0.487	0.50	0.000	38.323	38.323	38.323	0.000
	MSH_BB	110	2.271	0.486	0.50	0.129	38.220	38.282	38.320	0.042
	MSH_UE	921	2.260	0.483	0.50	0.121	38.218	38.263	38.307	0.032
	GS	563554	2.260	0.480	0.50	-	-	38.304	-	-
Aver.	TLUEGO_BB	226	-	-	-	0.015	15.478	15.478	15.479	0.000
	TLUEGO_UE	891	-	-	-	0.009	15.478	15.478	15.479	0.001
	MSH_BB	258	-	-	-	1.164	15.350	15.413	15.453	0.038
	MSH_UE	1091	-	-	-	0.516	15.290	15.409	15.469	0.067
	GS	490338	-	-	-	-	-	15.445	-	-

Table 3.3: Results for the problems with $i_{\max} = 15$. TLUEGO_BB ($\epsilon_1 = \epsilon_2 = 0.0001$), TLUEGO_UE, MSH_BB and MSH_UE (with 150 starting points), and GS.

			Best Solution			Max	Objective Function			
($j_{\max}$,k)	Algorithm	$Time$	x_1	y_1	α_1	$Dist$	Min	Av	Max	Dev
(2,0)	TLUEGO_BB	2714	4.580	6.183	4.99	0.016	45.533	45.593	45.637	0.042
	TLUEGO_UE	2991	4.580	6.184	4.99	0.030	45.472	45.541	45.592	0.056
	MSH_BB	2169	4.584	6.189	4.92	2.210	22.210	34.739	44.373	8.146
	MSH_UE	2652	4.581	6.185	4.32	4.964	24.210	32.102	40.052	5.405
	GS	876818	4.600	6.200	5.00	-	-	42.191	-	-
(2,1)	TLUEGO_BB	1324	7.066	7.225	4.93	0.110	61.041	61.049	61.055	0.006
	TLUEGO_UE	1957	7.066	7.225	4.89	0.105	61.044	61.045	61.050	0.002
	MSH_BB	1619	7.229	7.414	4.63	0.781	35.451	39.567	51.214	5.876
	MSH_UE	2630	7.072	7.231	4.78	1.148	49.804	55.859	60.568	3.566
	GS	745640	7.080	7.240	4.75	-	-	59.912	-	-
(5,0)	TLUEGO_BB	915	0.000	2.159	0.50	0.002	-10.847	-10.847	-10.846	0.000
	TLUEGO_UE	1579	0.000	2.159	0.50	0.002	-10.847	-10.847	-10.846	0.000
	MSH_BB	1503	0.000	2.158	0.50	0.064	-10.882	-10.867	-10.849	0.012
	MSH_UE	2163	0.002	2.158	0.50	0.028	-10.862	-10.853	-10.847	0.005
	GS	860059	0.020	2.140	0.50	-	-	-10.881	-	-
(5,1)	TLUEGO_BB	1704	10.000	8.657	0.50	0.000	13.492	13.492	13.492	0.000
	TLUEGO_UE	1495	10.000	8.657	0.50	0.000	13.492	13.492	13.492	0.000
	MSH_BB	2072	10.000	8.658	0.50	0.039	13.469	13.481	13.489	0.008
	MSH_UE	2322	9.999	8.657	0.50	0.002	13.480	13.488	13.492	0.005
	GS	844241	9.300	6.600	0.50	-	-	13.430	-	-
(5,2)	TLUEGO_BB	1957	6.152	2.006	2.47	0.085	46.736	46.739	46.741	0.002
	TLUEGO_UE	2251	6.152	2.006	2.51	0.521	46.502	46.666	46.732	0.085
	MSH_BB	2387	6.146	2.000	2.47	0.801	44.630	45.947	46.620	0.701
	MSH_UE	2793	6.149	2.010	2.73	0.671	45.249	45.911	46.389	0.542
	GS	762229	6.140	2.000	2.25	-	-	46.388	-	-
(10,0)	TLUEGO_BB	425	0.526	0.000	0.50	0.001	-10.283	-10.283	-10.283	0.000
	TLUEGO_UE	2135	0.526	0.000	0.50	0.003	-10.283	-10.283	-10.283	0.000
	MSH_BB	651	0.526	0.000	0.50	0.016	-10.289	-10.285	-10.283	0.002
	MSH_UE	3650	0.526	0.000	0.50	0.001	-10.284	-10.283	-10.283	0.001
	GS	1237873	0.520	0.000	0.50	-	-	-10.295	-	-
(10,2)	TLUEGO_BB	627	9.623	9.436	0.50	0.001	24.958	24.959	24.959	0.000
	TLUEGO_UE	1692	9.651	9.440	0.50	0.027	24.960	24.962	24.965	0.002
	MSH_BB	959	9.633	9.437	0.50	0.377	24.856	24.896	24.953	0.035
	MSH_UE	2285	9.615	9.438	0.50	0.281	24.854	24.911	24.948	0.035
	GS	963392	9.880	9.460	0.50	-	-	24.852	-	-
(10,4)	TLUEGO_BB	942	0.486	4.980	2.94	0.544	62.201	62.334	62.414	0.094
	TLUEGO_UE	1259	0.486	4.980	3.32	0.702	62.169	62.273	62.402	0.074
	MSH_BB	1449	0.481	4.982	3.31	2.302	57.799	59.881	62.081	1.794
	MSH_UE	2153	0.483	4.977	2.78	3.298	51.366	56.898	62.045	4.034
	GS	1021566	0.480	4.980	2.75	-	-	61.836	-	-
Aver.	TLUEGO_BB	1326	-	-	-	0.095	29.104	29.130	29.146	0.018
	TLUEGO_UE	1920	-	-	-	0.174	29.064	29.106	29.138	0.027
	MSH_BB	1601	-	-	-	0.824	22.156	24.670	27.700	2.072
	MSH_UE	2581	-	-	-	1.299	23.477	26.004	28.296	1.699
	GS	913977	-	-	-	-	-	28.429	-	-

Table 3.4: Results for the problems with $i_{\max} = 25$. TLUEGO_BB ($\epsilon_1 = \epsilon_2 = 0.0001$), TLUEGO_UE, MSH_BB and MSH_UE (with 200 starting points), and GS.

			Best Solution			Max	Objective Function			
($j_{\max}$,k)	Algorithm	$Time$	x_1	y_1	α_1	$Dist$	Min	Av	Max	Dev
(2,0)	TLUEGO_BB	14473	5.936	5.663	2.44	0.138	-9.998	-9.986	-9.972	0.010
	TLUEGO_UE	5557	5.936	5.663	2.45	0.222	-9.996	-9.985	-9.971	0.010
	MSH_BB	17699	6.051	5.644	2.72	8.323	-12.840	-12.642	-11.859	0.391
	MSH_UE	7242	5.939	5.661	2.23	8.203	-12.836	-11.499	-10.250	1.094
	GS	3003349	5.940	5.660	2.25	-	-	-10.276	-	-
(2,1)	TLUEGO_BB	8632	2.840	4.738	4.96	0.051	77.850	78.006	78.297	0.152
	TLUEGO_UE	6377	2.840	4.738	5.00	0.043	78.297	78.516	78.654	0.125
	MSH_BB	8080	2.839	4.726	4.98	1.729	28.929	51.910	76.424	19.328
	MSH_UE	7180	2.852	4.737	4.94	3.679	21.223	63.114	75.007	20.981
	GS	3003750	2.840	4.720	5.00	-	-	75.285	-	-
(5,0)	TLUEGO_BB	8295	2.643	1.097	4.98	0.000	27.917	27.917	27.917	0.000
	TLUEGO_UE	9483	2.645	1.096	4.98	0.005	27.917	28.127	28.267	0.172
	MSH_BB	9433	3.711	2.168	4.99	1.791	12.304	17.132	21.203	2.929
	MSH_UE	11058	2.640	1.099	4.99	2.337	6.922	15.253	27.402	7.428
	GS	3003816	3.720	2.160	4.50	-	-	21.678	-	-
(5,1)	TLUEGO_BB	10993	7.346	9.574	4.39	0.740	16.347	16.455	16.614	0.090
	TLUEGO_UE	7153	7.346	9.574	4.61	0.634	16.422	16.561	16.666	0.093
	MSH_BB	12956	7.042	9.555	4.74	2.520	-0.745	6.598	13.342	4.769
	MSH_UE	9112	7.265	9.583	4.13	1.952	0.359	8.730	12.275	4.303
	GS	3003513	7.340	9.580	4.00	-	-	15.153	-	-
(5,2)	TLUEGO_BB	11834	9.502	4.853	2.34	0.441	48.553	48.664	48.776	0.096
	TLUEGO_UE	16579	9.502	4.853	2.37	0.393	48.478	48.706	48.779	0.115
	MSH_BB	15717	9.497	4.851	2.03	5.111	46.868	47.116	48.106	0.495
	MSH_UE	20315	10.000	0.000	0.50	0.000	47.698	47.698	47.698	0.000
	GS	3004494	9.980	0.300	0.50	-	-	47.459	-	-
(10,0)	TLUEGO_BB	7825	1.008	7.438	5.00	0.073	31.359	31.580	31.734	0.180
	TLUEGO_UE	6510	1.008	7.438	5.00	0.099	31.223	31.524	31.734	0.204
	MSH_BB	9336	0.999	7.428	4.98	1.490	18.745	23.831	29.913	3.574
	MSH_UE	7495	2.206	8.129	4.93	1.145	23.221	26.986	30.718	2.655
	GS	3003878	1.020	7.420	5.00	-	-	28.702	-	-
(10,2)	TLUEGO_BB	4826	9.865	8.238	4.98	0.032	56.181	56.315	56.414	0.109
	TLUEGO_UE	6532	9.865	8.238	4.99	0.021	56.225	56.390	56.521	0.095
	MSH_BB	5199	9.545	7.674	4.84	1.288	43.832	47.754	54.446	3.663
	MSH_UE	6660	9.866	8.238	4.38	3.113	36.785	46.307	52.232	5.220
	GS	3003775	9.880	8.240	5.00	-	-	50.801	-	-
(10,4)	TLUEGO_BB	8884	7.675	3.264	4.22	0.013	70.718	70.729	70.738	0.009
	TLUEGO_UE	7880	7.675	3.264	4.24	0.061	70.730	70.737	70.741	0.004
	MSH_BB	10296	7.669	3.265	4.17	0.585	67.679	68.935	70.198	0.911
	MSH_UE	10223	7.661	3.255	3.92	1.727	66.780	68.119	69.594	1.090
	GS	3003775	7.460	3.040	4.00	-	-	69.442	-	-
Aver.	TLUEGO_BB	9470	-	-	-	0.186	39.866	39.960	40.065	0.081
	TLUEGO_UE	8259	-	-	-	0.185	39.912	40.072	40.174	0.102
	MSH_BB	11090	-	-	-	2.855	25.597	31.329	37.722	4.508
	MSH_UE	9911	-	-	-	2.769	23.769	33.088	38.084	5.346
	GS	3003794	-	-	-	-	-	37.280	-	-

Table 3.5: Results for the problems with $i_{\max} = 50$. TLUEGO_BB ($\epsilon_1 = \epsilon_2 = 0.0001$), TLUEGO_UE, MSH_BB and MSH_UE (with 250 starting points), and GS.

TLUEGO_BB and MSH_BB, by 12.79% and 10.63%, respectively. These results are also consistent with the ones showed in Subsection 2.3.4, where it was observed that the increase of requirements for iB&B with the size of the problem was greater than for UEGO.

Focusing now on the strategies proposed to solve the current centroid problem, it can be stated that TLUEGO (in any of its versions) is the algorithm providing the best results. Its average objective function values (see column 'Av' in tables 3.3, 3.4 and 3.5) are always higher than the ones given by both MSH and GS. It is also important to highlight that the minimum objective function value found by TLUEGO in the ten runs is always better than the average values obtained by both MSH and GS (see columns 'Min' and 'Av').

In addition to this, TLUEGO is also the most reliable algorithm, in the sense that it usually attains the same solution in all the runs, whereas MSH is more erratic, and may provide different solutions in each run (see the values of '$MaxDist$' and 'Dev').

MSH has been designed to check whether a random search is enough to find the global optimum of the centroid problem introduced in this paper. But as the results show, it is necessary to direct the search through the whole space, as TLUEGO does. In only four problems (those with settings $(15, 10, 0)$, $(15, 10, 2)$, $(25, 5, 1)$ and $(25, 10, 0)$) does MSH find the same best solution as TLUEGO (see columns 'Best Solution'). Furthermore, in none of the 24 problems the best objective function value found by MSH in the ten runs is better than the corresponding average value of TLUEGO (see columns 'Max' and 'Av'). Comparing MSH to GS, only in 10 out of the 24 problems is the average value of MSH greater than the objective value obtained by GS.

GS is rather time-consuming. Moreover, there is no guarantee that GS can find a good approximation to the global optimum. If the objective function value decreases dramatically in a small neighborhood around the global optimum and the grid is not dense enough, the second finer grid can focus around a local optimum. Something similar can happen when a local optimum exists whose objective value is close to the global optimum value and the grid is not fine enough. The risk of failure is even higher in the presence of constraints, as happens in our centroid problem, since it may occur that the global optimum is surrounded (in part) by infeasible areas, and the grid may not have a feasible point near the global optimum. Of course, the finer the grids, the higher the possibilities for the method to find the optimum, but one never knows how

		Max	Objective Function			
Algorithm	$Time$	$Dist$	Min	Av	Max	Dev
TLUEGO_BB	3674	0.099	28.149	28.189	28.230	0.033
TLUEGO_UE	3690	0.123	28.151	28.219	28.264	0.043
MSH_BB	4316	1.614	21.034	23.804	26.958	2.206
MSH_UE	4528	1.528	20.845	24.834	27.283	2.371
GS	1469370	-	-	27.052	-	-

Table 3.6: Average results considering all the problems ($i_{\max} = 15, 25, 50$).

small the distance between two adjacent points in the grid should be, and regardless how small that distance is, it may still happen that the search does not reach the global optimum. In fact, GS has been used only as a safeguard to check the goodness of TLUEGO and MSH, and also because it allows to study the difficulty of the problem at hand and to draw the graphs of the objective function projected in both the location and the quality spaces.

Finally, to have an overall view of the algorithms' behavior, average values when considering all the problems are presented in Table 3.6. As can be observed, similar conclusions can be inferred, i.e. TLUEGO is both the algorithm giving the best and most robust results, and this using the least computational time. On average, MSH provides the worst objective function value (see column Av) and different runs may provide very different objective values. GS is much more time-consuming and is not able to find the global optimum. Regarding the quality of the solutions, it seems that TLUEGO_UE and MSH_UE obtain better average results than their corresponding counterparts executed with iB&B.

To illustrate the algorithms' behavior, the best solution obtained by the different algorithms for the problem with setting $(50, 5, 0)$ are depicted in Figure 3.3, projected onto the two-dimensional spaces (see also Table 3.7). In that figure, the black squares (■) correspond to the locations of the existing follower's facilities, the green symbol + gives the best solution found by TLUEGO_BB in the ten runs, and the blue star ✳ is the one obtained by TLUEGO_UE; the green and red signs × give the best solutions provided by MSH_BB and MSH_UE, respectively. Finally, the red plus sign + represents the solution obtained by the GS algorithm. Light yellow ovals represent the forbidden areas around the existing demand points, which are at the center of those

Algorithm	*Time*	Best Solution x_1	y_1	α_1	*Max Dist*	Objective Function *Min*	*Av*	*Max*	*Dev*
TLUEGO_BB	8295	2.643	1.097	4.98	0.000	27.917	27.917	27.917	0.000
TLUEGO_UEGO	9483	2.645	1.096	4.98	0.005	27.917	28.127	28.267	0.172
MSH_BB	9433	3.711	2.168	4.99	1.791	12.304	17.132	21.203	2.929
MSH_UEGO	11058	2.640	1.099	4.99	2.337	6.922	15.253	27.402	7.428
GS	3003816	3.720	2.160	4.50	-	-	21.678	-	-

Table 3.7: Results for the problem with setting $(50, 5, 0)$.

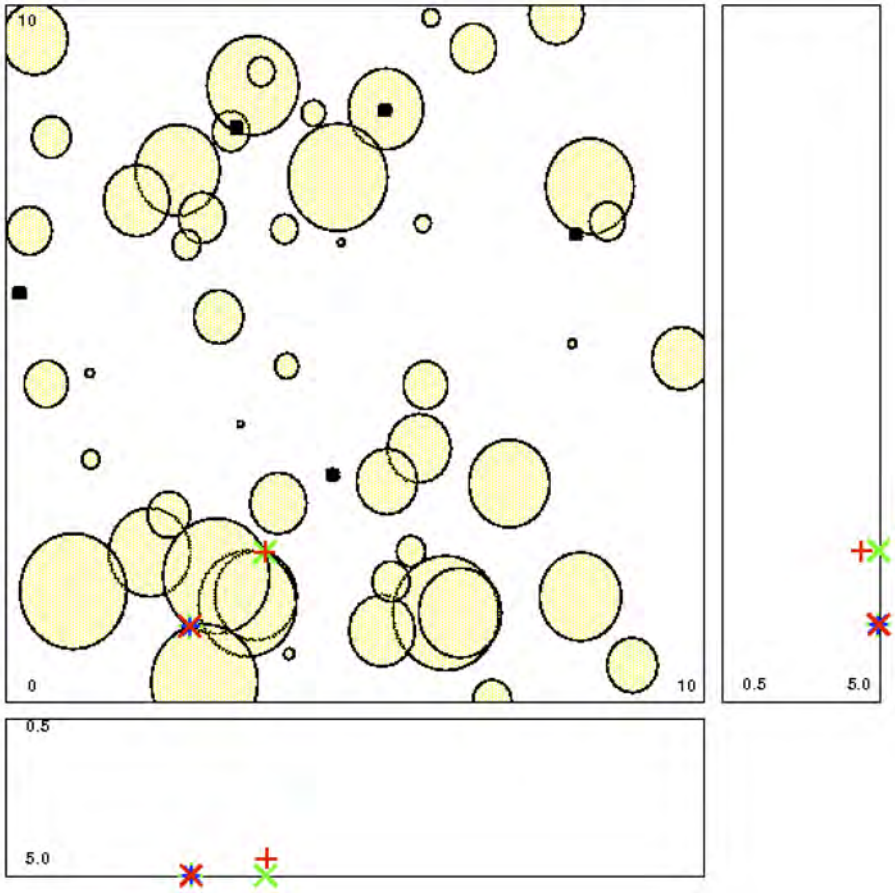

Figure 3.3: Example with $i_{\max} = 50, j_{\max} = 5, k = 0$. TLUEGO_BB = + (green), TLUEGO_UEGO = $\ast$ (blue), MSH_BB = $\times$ (green) MSH_UEGO = $\times$ (red) and GS = + (red).

ovals (the greater the oval, the greater the purchasing power at the demand point). Notice that in this problem $k = 0$, i.e., there are no existing facilities belonging to the leader's chain.

As can be seen in this example, the optimal solution is at the intersection of two forbidden regions, as found by both TLUEGO_BB and TLUEGO_UE. The solution provided by MSH_UE is around that area, but it focuses on a local maximum whose objective value is close to the optimal one (see 'Max' column in Table 3.7). Also notice that the solutions provided by GS and MSH_BB are quite close, but they yield very different objective values, showing that the objective function can locally be quite steep. Thus, this example clearly shows that GS is not a good strategy, since it does not allow a proper approximation to the optimal point. Additionally, MSH (in any of its versions) may get trapped in a local optimum, since the goodness of that algorithm depends on how close the starting points are with respect to the optima.

3.3 Influence of the fuse process in the creation procedure

As was mentioned before, TLUEGO is based on the algorithm UEGO_cent.SASS described in [133]. In the creation procedure, for every species in the list, a set of possible new candidates is computed, fused and evaluated in order to find new promising species, thereby increasing the species-list. Notice that each species generates new ones by itself, independent of the remaining ones.

An important issue to take into account is that the evaluation of a single species in TLUEGO requires intensive computational effort, since it involves the execution of another expensive optimization algorithm (UEGO or iB&B) to obtain the optimal location of the follower (by solving the corresponding medianoid problem). For this reason TLUEGO was designed to maintain a small-size species-list, by including a *'fuse'* process just after the creation of candidate solutions, therefore, only the resulting ones are evaluated.

However, it is known that larger species-list sizes help to explore the search space deeply and then obtain better solutions. With this aim, in this section, new creation procedures are proposed, where the fuse process is relaxed in part (instead of fusing two species whenever the distance between their centers is smaller than $2R_t$, the new thresholds R_t, $R_t/2$ or 0 are used). In what follows, only TLUEGO_UE will be used, since it can solve larger instances. It will simply be denoted by TLUEGO.

$i_{\max}$	50			100		
$j_{\max}$	2	5	10	2	5	10
k	0,1	0,1,2	0,2,4	0,1	0,1,2	0,2,4

Table 3.8: Settings of the test problems.

For the analysis at hand, only small and medium size problems have been considered, i.e. $i_{\max} = 50, 100$. For every setting in Table 3.8, a problem was generated by randomly choosing its parameters uniformly within the intervals presented in Appendix. In all the problems, $FR_1 = FR_2 = ([0, 10], [0, 10])$ and $\alpha_1, \alpha_2 \in [0.5, 5]$. They have been solved in the Arabi supercomputer, using a single core of the nodes with 8GB memory (one problem at a time).

Since each run of TLUEGO may provide a different solution, for every variant, each problem has been solved ten times and average values have been computed. Tables 3.9 and 3.10 show the results obtained by the algorithms for the problems with $i_{\max} = 50$ and $i_{\max} = 100$, respectively. The first column gives the setting of the problem. The second one indicates the threshold value used in the fuse process. In the column labeled '$Time$', the average time in the ten runs (in seconds) is showed. The $MaxDist$ column gives the maximum Euclidean distance (considering the three variables of the problem) between any pair of solutions provided by the algorithm in the ten runs, which provides an idea of how far the solutions given by the algorithm in different runs can be. The average objective function value (column Π_1) in the ten runs and the corresponding standard deviation (column Dev) are given next. Column $Dif\Pi_1$ shows the relative improvement in the objective function value between the solution obtained by the algorithms when a threshold different from $2R_t$ is used as compared to the result obtained when using that threshold, and it is computed as $Dif\Pi_1 = \frac{\Pi_1^{(other)} - \Pi_1^{(2R_t)}}{\Pi_1^{(2R_t)}} \cdot 100$. The final column shows the relative difference between the solutions, and is computed as

$$DifSol = \frac{\|nf_1^{(other)} - nf_1^{(2R_t)}\|_2}{\max\{\|nf_1^{(2R_t)} - (x_1, y_1, \alpha_1)\| : (x_1, y_1, \alpha_1) \in FR_1 \times [0.5, 5]\}} \cdot 100.$$

The last four lines in the tables give the average values when considering all the problems.

$(j_{\max}, k)$	threshold	$Time$	$MaxDist$	Π_1	Dev	$Dif\Pi_1$	$DifSol$
(2,0)	$2R_t$	5587	0.509	49.311	0.027	-	-
	R_t	8839	0.320	49.390	0.020	0.159	0.822
	$R_t/2$	9281	0.261	49.423	0.005	0.226	1.263
	0	13282	0.197	49.443	0.006	0.266	0.424
(2,1)	$2R_t$	4871	0.159	255.137	0.732	-	-
	R_t	7136	0.068	257.168	0.267	0.796	1.812
	$R_t/2$	7632	0.023	257.831	0.085	1.056	2.440
	0	10042	0.007	257.974	0.021	1.112	2.568
(5,0)	$2R_t$	9408	0.317	171.041	0.956	-	-
	R_t	15242	0.118	172.414	0.322	0.803	1.545
	$R_t/2$	15888	0.099	173.201	0.178	1.263	2.622
	0	20804	0.049	173.878	0.075	1.659	3.725
(5,1)	$2R_t$	10154	0.106	115.437	0.164	-	-
	R_t	18190	0.114	115.970	0.131	0.462	1.135
	$R_t/2$	18084	0.084	116.344	0.080	0.786	2.074
	0	21348	0.140	116.792	0.085	1.174	3.593
(5,2)	$2R_t$	17093	0.890	95.705	0.086	-	-
	R_t	26150	0.664	96.445	0.050	0.773	0.254
	$R_t/2$	29045	0.217	97.011	0.040	1.365	3.991
	0	34462	0.402	98.971	0.031	3.412	3.016
(10,0)	$2R_t$	11623	0.922	137.950	0.829	-	-
	R_t	16800	0.788	139.586	0.431	1.187	2.335
	$R_t/2$	18658	0.109	140.934	0.391	2.163	3.531
	0	21643	0.057	141.850	0.202	2.827	3.717
(10,2)	$2R_t$	14212	0.784	198.422	1.756	-	-
	R_t	24459	0.031	201.940	0.124	1.773	2.084
	$R_t/2$	25590	0.023	202.602	0.100	2.107	2.274
	0	30152	0.017	202.807	0.080	2.210	2.344
(10,4)	$2R_t$	14997	0.469	163.525	0.073	-	-
	R_t	24696	0.351	164.015	0.070	0.300	4.507
	$R_t/2$	25307	0.215	165.024	0.024	0.917	0.714
	0	31449	0.212	166.298	0.013	1.696	4.131
Aver	$2R_t$	10993	0.520	148.316	0.578	-	-
	R_t	17689	0.307	149.616	0.177	0.782	1.812
	$R_t/2$	18686	0.129	150.296	0.113	1.235	2.364
	0	22898	0.135	151.002	0.064	1.794	2.940

Table 3.9: Effectiveness evaluation of the fuse process in TLUEGO (sequential algorithm) for problems with $i_{\max} = 50$ demand points.

$(j_{\max}, k)$	threshold	$Time$	$MaxDist$	Π_1	Dev	$Dif\Pi_1$	$DifSol$
(2,0)	$2R_t$	20567	0.443	173.455	2.209	-	-
	R_t	34793	0.095	177.655	0.343	2.422	3.210
	$R_t/2$	36818	0.028	179.286	0.101	3.362	4.544
	0	43798	0.033	180.223	0.116	3.902	5.381
(2,1)	$2R_t$	28921	0.969	209.147	1.563	-	-
	R_t	43661	0.108	213.918	0.374	2.281	3.515
	$R_t/2$	47264	0.039	216.137	0.123	3.342	4.986
	0	56301	0.001	217.040	0.023	3.774	5.755
(5,0)	$2R_t$	31206	0.359	135.318	1.151	-	-
	R_t	48181	0.165	137.661	0.486	1.731	2.358
	$R_t/2$	53020	0.083	138.694	0.200	2.495	3.507
	0	59954	0.022	139.208	0.054	2.875	4.110
(5,1)	$2R_t$	24066	0.583	220.752	2.856	-	-
	R_t	41364	0.179	225.347	0.736	2.082	3.301
	$R_t/2$	44739	0.046	228.335	0.230	3.435	5.610
	0	51167	0.005	228.662	0.022	3.583	5.811
(5,2)	$2R_t$	33344	0.383	179.378	1.152	-	-
	R_t	56804	0.113	181.226	0.258	1.030	2.100
	$R_t/2$	63904	0.170	182.576	0.461	1.783	3.845
	0	73450	0.001	183.258	0.002	2.163	4.788
(10,0)	$2R_t$	38772	1.101	165.054	3.058	-	-
	R_t	62530	0.276	170.901	1.362	3.543	3.921
	$R_t/2$	67270	0.159	177.146	0.897	7.326	3.999
	0	74407	0.031	178.971	0.161	8.432	4.627
(10,2)	$2R_t$	38916	1.094	114.965	1.847	-	-
	R_t	65372	0.163	119.828	0.154	4.230	5.938
	$R_t/2$	69543	0.510	120.275	0.099	4.618	8.615
	0	79906	0.333	120.474	0.042	4.791	10.929
(10,4)	$2R_t$	40438	1.111	220.841	2.104	-	-
	R_t	64295	0.067	240.192	0.206	8.763	9.425
	$R_t/2$	72898	0.030	243.230	0.066	10.138	12.875
	0	84776	0.019	244.570	0.043	10.745	14.819
Aver	$2R_t$	32029	0.755	177.364	1.992	-	-
	R_t	52125	0.146	183.341	0.490	3.260	4.221
	$R_t/2$	56932	0.133	185.710	0.272	4.562	5.998
	0	65470	0.056	186.551	0.058	5.033	7.027

Table 3.10: Effectiveness evaluation of the fuse process in TLUEGO (sequential algorithm) for problems with $i_{\max} = 100$ demand points.

As can be seen, the CPU time increases as the threshold decreases, and when this is set to 0, the time is (a bit more than) double as compared to the $2R_t$ case. The algorithm also becomes more robust, in the sense that the runs give solutions with a more similar objective function value (see the decrease in the figures in columns *Dev*). And the quality of the solution gets better, as its average objective function value gets better (see columns Π_1). Concerning the relative improvement in the objective function value, whereas for the problems with $i_{\max} = 50$ demand points are on average a moderate 1.794% when the threshold is set to 0, for the problems with $i_{\max} = 100$ it attains a significant 5.033%. Furthermore, for some particular instances the progress may be more than 10%. This clearly shows that the smaller the threshold, the better the solutions are, although this is at the cost of increasing the CPU time and the memory requirements.

3.4 High performance computing

According to the studies showed in the previous secticon, the computational time employed by TLUEGO for solving small size problems was very high, and this despite the inclusion of a *fuse* process in the creation procedure to reduce the size of the species-list in the algorithm (see tables 3.9 and 3.10). This clearly suggests that a parallelization of the algorithm is needed, especially if real problems, with more demand points, must be solved.

In the following, some details about TLUEGO which may affect the efficiency of the parallel algorithms are described.

- The first aspect to highlight is that there is no relationship among species. This means that a single species can create a new offspring and evolve to the local or global optima without participation of the remaining ones. Therefore, there exists an intrinsic parallelism, which can be exploited by dividing the species-list among the available processors.

- The second one is related to the *Selection* procedure. As mentioned above (see also [131]), at each iteration TLUEGO calls the *Selection* mechanism twice, just after the *Create_species* and *Optimize_species* procedures. The *Selection* procedure requires the knowledge of the whole species-list to be able to measure the

distances among all the species for the fuse process, and therefore the processors must interchange information. This may be understood as synchronization points, which may reduce the efficiency of the parallel version. On the other hand, the *Selection* method itself is quite fast, since it only implies the computation of distances between species. It means that the parallelization of the *Selection* procedure may be counterproductive, since the parallelism overheads may be higher than the computational cost saved. To deal with the drawbacks imposed by the *Selection*, a good load balancing must be designed.

- The third one is that both the *Create_species* and the *Optimize_species* procedures are time-consuming, which makes them suitable for being run in parallel.

In the following subsections, three programming paradigms for the parallelization of TLUEGO are designed. More specifically, a pure message passing paradigm (Subsection 3.4.1), a pure shared memory programming model (Subsection 3.4.2) and a hybrid one which combines message passing with shared memory (Subsection 3.4.3). Finally, in Subsection 3.4.4 their efficiency and effectiveness are analyzed and compared.

3.4.1 Pure message passing programming for TLUEGO: PMP_TLUEGO

The first parallel approach of TLUEGO has been designed to be executed in a multicomputer, and hence the programming language is based on message-passing mechanisms. A master-slave technique has been implemented considering the characteristics of these kinds of optimization problems with a time-consuming objective function. In [138], the counterpart of the current problem was considered, i.e. the facility location and design (1|1)-centroid problem with exogenous demand was solved via parallel algorithms. In that work, several parallel strategies were designed and analyzed. The results showed that the efficiency of the master-slave method outperformed the other proposals. That parallel method has been adapted to the problem at hand.

Algorithm 8 depicts the main structure of the master-slave model. Next, the main details are summarized (the interested reader is referred to [138] for a deeper description of the parallel model).

Algorithm 8: PMP_TLUEGO

1: Init_species-list
2: **for** $t = 1$ to $t_{\max}$ **do**
3: Create_species_paral
4: Selection
5: Optimize_species_paral
6: Selection

In our particular master-slave model, the master processor executes TLUEGO sequentially. The parallelism comes from the simultaneous evaluation of the new candidate solutions in the creation function, and from the concurrent execution of the local search procedure (see [131]). Therefore, new creation and optimization procedures have been designed to cope with the parallel model. These new procedures will be called *Create_species_paral* and *Optimize_species_paral*.

Basically, when the *Create_species_paral* takes place, the master obtains a new offspring of candidate solutions for the leader's facility. The evaluation of the objective function Π_1 at those candidate solutions is carried out in a parallel way. To this aim, the master processor divides the list of candidate solutions by the number of processors and delivers the resulting sublists among them all (including itself). Each processor receives a species sublist from the master and evaluates Π_1 at each of its elements. Notice that the amount of information involved in these communications is pretty small: only the location and the quality of the leader's facility (the center of the species) need to be sent.

Optimize_species_paral behaves similar to the previous procedure. Again, the master divides the species list among the processors and distributes the resulting sublists. Nevertheless, now, each processor executes the local search method SASS+WLWv (see Subsection 3.2.2) to every species in its sublist.

Similar to TLUEGO, the parallel version PMP_TLUEGO includes a selection mechanism just after the creation and optimization procedures (see Steps 4 and 6 of Algorithm 8). For the reasons expound previously, the execution of these procedures are only carried out by a single processor, the master.

It is important to mention that the interchange of information among processors has been carried out through MPI [69]. Processes are written in C++, and communications and synchronizations are carried out by calling functions from the MPI library.

3.4.2 Shared memory programming for TLUEGO: SMP_TLUEGO

The second parallel approach of TLUEGO is devised to be executed in a multiprocessor, hence, shared memory programming is considered. Contrary to the previous parallel strategy, no messages are required to communicate processors, though a mechanism to coherently share memory data is necessary. For the implementation of the problem at hand, OpenMP has been selected, since it is a portable and scalable model, and gives programmers a simple and flexible interface for developing parallel applications.

Concerning the parallel model, it can be considered a *pseudo* master-slave technique. OpenMP includes mechanisms to distribute the species list among the different processors without the existence of a master processor. In this way there does no exist a master processor which globally controls the algorithm and manages the species list. It can be done in parallel by all the processors. Nevertheless, it is still necessary that a single processor be in charge of applying the *Selection* procedure and updating the species list that will be accessible to all processors. Accordingly, the parallelism is applied to the evaluation of the new candidate solutions in the creation function and to the local search procedure. Therefore, new creation and optimization procedures have also been designed. The structure of the proposed algorithm is similar to Algorithm 8, although in this case, it cannot be considered a master-slave technique in the sense that unlike PMP_ TLUEGO, in this case, no processor has unidirectional control over the remaining ones. In the following, the main ideas of the SMP_TLUEGO algorithm are summarized.

The parallel algorithm developed in this subsection considers that the species-list is stored in shared memory. When the *Create_species_paral* is executed, each processor picks up a new single species and evaluates it. Once a particular processor has finished that task, it collects another species. This loop is repeated until all the new offspring are evaluated. Notice that mutual exclusion is not needed, since each processor accesses different memory areas. Additionally, note that no processor makes decisions about what species must be evaluated by which processor, as happens in the previous version. On the contrary, all the processors remain at the same hierarchical level.

Optimize_species_paral procedure differs from the *Create_species_paral* method in the task carried out by each processor. Instead of only evaluating the species, it applies the local search procedure. Notice that the number of function evaluations required to

optimize a single species and hence, the computational load assumed by each processor, may be quite different. The way the species are distributed among the processors helps to balance the computational burden and to reduce the waiting time of the processors.

3.4.3 Hybrid parallel programming for TLUEGO: HPP_TLUEGO

This last parallel version of TLUEGO has been conceived to be executed in a multicomputer where each node is a multiprocessor (see Figure 1.6), like for instance, Arabi. The parallel programming combines message-passing mechanisms among nodes with shared memory parallelization inside each node. MPI and OpenMP have been considered to implement the parallel version. The parallel model links a *coarse-grain model* with a pseudo *master-slave strategy*.

For the problem at hand, each node executes TLUEGO. The species-list size and the total number of function evaluations for the whole optimization process are internally divided by the number of nodes, N. Concerning the migration procedure, two types of nodes (collectors and workers) are considered. Half of the nodes act as collectors and the other half as workers. At each communication, each node behaves either as a worker (sender) or as a collector (receiver), although, they interchange their roles at the next communication. The nodes are supposed to be connected in a ring topology and run independent of the remaining ones. In a communication stage, node i is a worker and sends its sublist to the next node *i+1* (collector) (see the left part of Figure 3.4). Node *i+1* fuses this list with its own sublist and distributes the resulting list between both nodes (see the right part of Figure 3.4). In the next communication stage, node i will be a collector and will receive a sublist from node *i-1*, while node *i+1* will be a worker and will send the sublist to the node *i+2* (see Figure 3.5). The migration process is carried out at the first half of the levels of the algorithm. The communications among nodes are implemented using MPI.

Notice that with the previous parallel strategy only, the computational resources inside each node are not fully exploited, since only a single processor would be used. To make use of the whole set of processors and improve the efficiency of the parallel version, the parallel algorithm SMP_TLUEGO is considered inside each node instead of the sequential version TLUEGO.

It is important to mention that parallel hybrid versions based on the master-slave

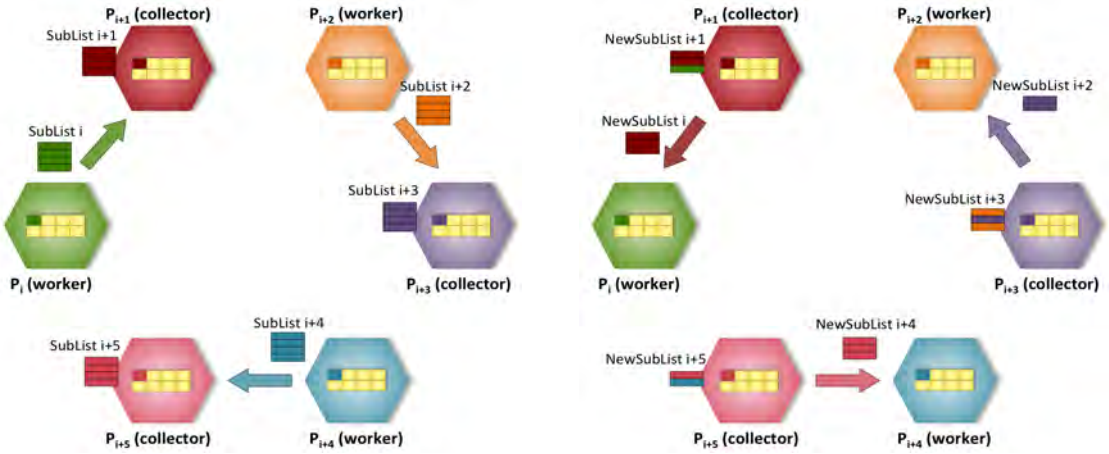

Figure 3.4: HPP_TLUEGO parallel strategy. Communication j.

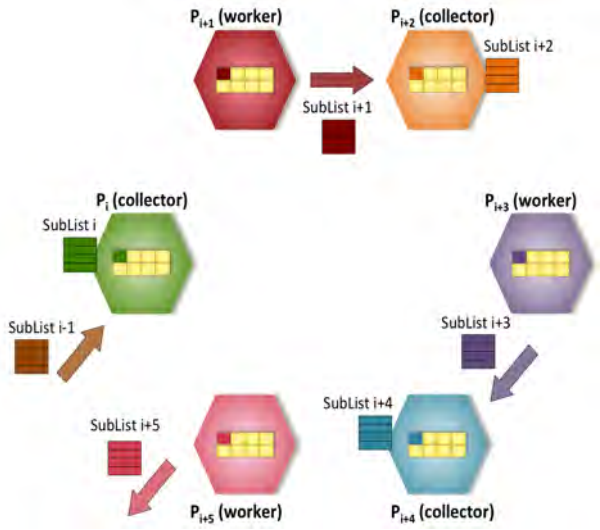

Figure 3.5: HPP_TLUEGO parallel strategy. Communication $j+1$.

strategy instead of the coarse grain paradigm as first hybridization level have also been implemented. Nevertheless, the achieved efficiencies were quite poor, since the overhead imposed by the communication costs increases. Notice that, following the idea of the master-slave model, a processor inside a node will act as master whereas the remaining ones will be slaves. This implies that the number of communications for this case will be multiplied by N_c, the number of processors inside each node, as compared to the SMP_TLUEGO parallel version.

3.4.4 Computational studies

All the computational studies have been run in Arabi, always considering nodes with 8GB (see Subsection 1.3.5). The algorithms have been implemented in C++.

In order to study the performance of the parallel algorithms, 24 different problems have been considered. In particular, the medium size problems ($i_{\max} = 100$) described in Table 3.8 and used in the Section 3.3 have been considered. In addition, for every setting of Table 3.11 a problem was generated by randomly choosing its parameters uniformly within pre-specified intervals presented in Appendix. In all those problems, the following choices were made: $\alpha_1, \alpha_2 \in [0.5, 5]$, and $FR_1, FR_2 = ([0, 25], [0, 25])$.

Notice that all the parallel versions of TLUEGO have been developed considering that the threshold used in the fuse process in the creation procedure has been set to 0, since it provides the better results.

So as to study the performance of PMP_TLUEGO, it has been considered that it runs in $N = 1, 2, 4, 8, 16, 32, 64$ nodes, using a single processor per node ($N_c = 1$). Concerning the SMP_TLUEGO performance analysis, it has been carried out in a single node with 8GB, then $N = 1$ and $N_c = 1, 2, 4, 8$. Finally, the performance of HPP_TLUEGO has also been studied by considering $N = 1, 2, 4, 8$ nodes, but now using its $N_c = 8$ processors, i.e., all the available resources in a node have been selected.

To measure the performance of the parallel implementations when using P processors, the *efficiency* metric has been used, $Eff(P) = \frac{T(1)}{P \cdot T(P)}$. Notice that $T(1)$ is the time employed by the sequential TLUEGO, and the value of P is given by the number N of nodes multiplied by the number N_c of processors used in the nodes.

Table 3.12 shows, for the problems with $i_{\max} = 100$ demand points, the average computing time (in secs.) and the mean efficiency $Eff(P)$ obtained by each parallel algorithm. Again, the algorithms have been run 10 times for each problem. As can be seen, PMP_ TLUEGO as well as SMP_TLUEGO have either optimal or near-optimal efficiency for up to $P = 8$ processors. The efficiency values decrease when the number

$i_{\max}$	500		
$j_{\max}$	2	15	25
k	0,1	0,5,10	0,7,15

Table 3.11: Settings of the test problems.

$i_{\max}$	Algorithm	N	N_c	P	$Time$	$Eff(P)$
100	PMP_TLUEGO	1	1	1	65470	-
		2	1	2	32774	1.00
		4	1	4	16507	0.99
		8	1	8	8934	0.92
		16	1	16	4861	0.84
		32	1	32	2920	0.70
		64	1	64	1800	0.57
	SMP_TLUEGO	1	1	1	65470	-
		1	2	2	32878	1.00
		1	4	4	16928	0.97
		1	8	8	8703	0.94
	HPP_TLUEGO	1	1	1	65470	-
		2	8	16	5013	0.82
		4	8	32	3047	0.67
		8	8	64	2006	0.51

Table 3.12: Efficiency results for problems with $i_{\max} = 100$.

of processors P increases. This tendency, which also appears in the HPP_TLUEGO parallel version, may be explained by the parallel overheads and the waiting time, which increase as the number of processors does, but so does also by the limited computational load. Notice that a particular problem has a fixed burden which cannot be divided and distributed constantly by larger values of P.

Apparently, in terms of efficiency, the HPP_TLUEGO parallel version is worse than PMP_TLUEGO, since the obtained values for $P = 16, 32, 64$ are smaller. Nevertheless, notice that HPP_TLUEGO fully exploits the resources at their disposal, while PMP_TLUEGO does not (it requires double nodes to obtain similar efficiency values).

Finally, the *scalability* of the proposed algorithms is analyzed. Broadly speaking, this concept can be defined as the ability of an algorithm to manage a growing amount of work. Table 3.13 shows the mean executing time and the average efficiency achieved by the three parallel methods for instances with $i_{\max} = 500$ demands points. As can be seen, all the proposed algorithms demonstrate increased performance by increasing the problem size, i.e. the efficiency improves with regards to the previous set of experiments (with $i_{\max} = 100$), see Table 3.12. On the other hand, the same tendency in the efficiency values can be observed. Again, the efficiency decreases as the number of processing elements increases. Those behaviors may be explained by the same arguments as before.

$i_{\max}$	Algorithm	N	N_c	P	$Time$	$Eff(P)$
500	PMP_TLUEGO	1	1	1	565358	-
		2	1	2	282562	1.00
		4	1	4	144663	0.98
		8	1	8	75520	0.94
		16	1	16	39194	0.90
		32	1	32	21699	0.81
		64	1	64	12290	0.72
	SMP_TLUEGO	1	1	1	565358	-
		1	2	2	283707	1.00
		1	4	4	143416	0.99
		1	8	8	73065	0.97
	HPP_TLUEGO	1	1	1	565358	-
		2	8	16	40751	0.87
		4	8	32	24953	0.71
		8	8	64	14773	0.60

Table 3.13: Efficiency results for problems with $i_{\max} = 500$.

3.5 Conclusions

In this study, a new (1|1)-centroid problem on the plane with endogenous demand has been introduced. In the problem, a chain (the leader) has to decide where to locate a new facility (and its quality) knowing that a competitor (the follower) will react by locating another facility. The demand is assumed to be endogenous, depending on the distances to and on the quality of the facilities. Three heuristics have been proposed for handling the problem, namely, a grid search procedure, a multistart method and an evolutionary algorithm. The computational studies have shown that the evolutionary algorithm TLUEGO provides the best results and is more robust than the other strategies. However, the computational time employed by TLUEGO for solving a problem with 50 demand points is in average more than 2.5 hours. This clearly suggests that a parallelization of the algorithm is needed, especially if real problems, with more demand points, are to be solved.

A threshold value used within the fuse process included in the creation procedure of TLUEGO has been investigated. It has been shown that the smaller the threshold, the better the results obtained, but at the cost of an increase in CPU time and memory requirements.

Furthermore, three parallel implementations of TLUEGO, devised to be executed in different architecture platforms, have been proposed. All of them exhibit good per-

formance behaviors, obtaining either optimal or near-optimal efficiency for up to 8 processors. Additionally, their scalability has been demonstrated by solving problems with different computational loads, and checking that the efficiency increases with the size of the problems.

Depending on the computational environment available to the decision-maker, the most suitable threshold and/or parallel algorithm should be selected. For instance, to take advantage of today's multicore personal computers, the shared memory programming implementation SMP_TLUEGO should be used, with a threshold equal to 0 (if memory allows). On the contrary, when working in a cluster as those available in most of today's supercomputing centers, the hybrid algorithm HPP_TLUEGO should be used, since it allows to fully exploit all the computer power available in the nodes used.

CHAPTER 4

Expanding a franchise: solving a planar bi-objective facility location and design problem

The literature on multi-objective competitive location models is rather scarce. This is in part due to the fact that single-objective competitive location problems are difficult to solve, and considering more than one objective makes the problem nearly intractable. As far as the author knows, [60, 62, 63, 153] seem to be the only references on the topic.

In this chapter, the bi-objective problem described in [63] is revisited. A franchise wants to increase its presence in a given geographical region by opening one new facility. Both the franchisor (the owner of the franchise) and the franchisee (the actual owner of the new facility to be opened) have the same objective: maximizing their own profit. However, the maximization of the profit obtained by the franchisor is usually in conflict with the maximization of the profit obtained by the franchisee.

As was mentioned in Chapter 1, solving a multi-objective problem means obtaining the whole efficient set. To the extent of the author's knowledge, only two exact general methods, namely, two interval branch-and-bound methods (see [62, 63]) have been proposed in literature with that purpose for the general nonlinear bi-objective problem (1.4). However, they are time consuming. Furthermore, they have large memory requirements, so that only small instances can be solved with them.

On the other hand, the use of (meta)heuristics may allow to obtain good approximations of the efficient set, even for big instances. In particular, genetic and evolutionary algorithms have proved to be good tools to cope with (1.4), see for instance the well-known NSGA-II [40] or SPEA2 [162] algorithms. This is due to their ability to find multiple efficient solutions in one single run. However, they are usually designed to obtain a finite set of points approximating the Pareto-front given a budget in the number

of function evaluations, regardless the CPU time needed for that. In this thesis, a new Fast and Efficient Multi-Objective Evolutionary Algorithm (FEMOEA), whose aim is to obtain a good fixed size approximation of the Pareto-front quickly, is presented. To this aim, a new method to improve the efficiency of points is proposed. A new stopping rule is also introduced in FEMOEA, which allows to stop the algorithm as soon as a good approximation of the Pareto-front is available, thus saving time as compared to other criteria in which a fixed number of iterations or function evaluations have to be performed.

The chapter is organized as follows. In Section 4.1, the bi-objective facility location and design problem is described. In Section 4.2, the new algorithm FEMOEA is introduced and applied to the bi-objective facility location problem. To show the performance of the new method, the quality of the approximations of the Pareto-front provided by the algorithm is analyzed in the computational studies (Subsection 4.2.6). High performance computing approaches are investigated in Section 4.3. Finally, in Section 4.4 the main conclusions are summarized.

4.1 The model

A franchise wants to increase its presence in a given geographical region by opening one new facility. Both the franchisor and the franchisee want to maximize their own profit.

In the model, the *demand* is supposed to be exogenous and concentrated at $i_{\max}$ demand points, whose locations $locd_i$ and buying power $\widehat{w}_i$ are known (notice that in this model the demand is exogenous, i.e., fixed). The location $locf_j$ and quality of the existing facilities are also known. As in the previous chapters, the demand points split their buying power among *all* the facilities proportionally to the *attraction* they feel for them (see [83, 96]). The attraction (or utility) function of a customer towards a given facility depends on the distance between the customer and the facility as well as on other characteristics of the facility which determine its *quality*. The location and the quality of the new facility are the variables of the problem.

The following notation will be used throughout this chapter:

Indices

i index of demand points, $i = 1, \ldots, i_{\max}$.

j index of existing facilities, $j = 1, \ldots, j_{\max}$.

Variables

z location of the new facility, $z = (x, y)$.

α quality of the new facility ($\alpha > 0$).

nf variables of the new facility, $nf = (z, \alpha)$.

Data

$locd_i$ location of the i-th demand point.

$\widehat{w}_i$ demand (or buying power) at $locd_i$.

$locf_j$ location of the j-th existing facility.

$d_{i,j}$ distance between $locd_i$ and $locf_j$.

$a_{i,j}$ quality of facility j as perceived by demand point i.

$g_i(\cdot)$ a non-negative non-decreasing function.

$\frac{a_{i,j}}{g_i(d_{ij})}$ attraction that demand point i feels for facility j.

γ_i weight of the quality of the new facility as perceived by demand point i.

k number of existing facilities that are part of the franchise (the first k of the $j_{\max}$ facilities are assumed to be in this category, $0 < k < j_{\max}$).

Miscellaneous

$d_i(z)$ distance between demand point i and the new facility nf.

$\frac{\gamma_i \alpha}{g_i(d_i(z))}$ attraction that demand point i feels for nf.

From the previous assumptions, the total market share attracted by the franchisor is

$$M(nf) = \sum_{i=1}^{i_{\max}} \widehat{w}_i \frac{\dfrac{\gamma_i \alpha}{g_i(d_i(z))} + \displaystyle\sum_{j=1}^{k} \frac{a_{i,j}}{g_i(d_{i,j})}}{\dfrac{\gamma_i \alpha}{g_i(d_i(z))} + \displaystyle\sum_{j=1}^{j_{\max}} \frac{a_{i,j}}{g_i(d_i(z))}}.$$

It is assumed that the operating costs for the franchisor pertaining to the new facility are fixed. In this way, the profit obtained by the franchisor is an increasing function of the market share that it captures. Thus, maximizing the profit obtained by the franchisor is equivalent to maximizing its market share. This will be the first objective of the problem.

The second objective of the problem is the maximization of the profit obtained by

the franchisee, to be understood as the difference between the revenues obtained from the market share captured by the new facility minus its operational costs. The market share captured by the new facility (franchisee) is given by

$$m(nf) = \sum_{i=1}^{i_{\max}} \widehat{w}_i \frac{\dfrac{\gamma_i \alpha}{g_i(d_i(z))}}{\dfrac{\gamma_i \alpha}{g_i(d_i(z))} + \displaystyle\sum_{j=1}^{j_{\max}} \frac{a_{i,j}}{g_i(d_{i,j})}}$$

and the profit is given by the following expression,

$$\pi(nf) = F(m(nf)) - G(nf),$$

where $F(\cdot)$ is a strictly increasing function which determines the expected sales (i.e., income generated) for a given market share $m(nf)$, and $G(nf)$ is a function which gives the operating costs of a facility located at z with quality α.

In our computational studies we have considered F to be linear and G to be separable, in the form $G(nf) = G^a(z) + G^b(\alpha)$, where $G^a(z) = \sum_{i=1}^{i_{\max}} \Phi_i(d_i(z))$, with $\Phi_i(d_i(z)) = \widehat{w}_i/((d_i(z))^{\phi_{i0}} + \phi_{i1})$, $\phi_{i0}, \phi_{i1} > 0$, and $G^b(\alpha) = e^{\frac{\alpha}{\alpha_0}+\alpha_1} - e^{\alpha_1}$, with $\alpha_0 > 0$ and α_1 given values (other possible expressions for $G(nf)$ can be found in [61]).

The problem considered is

$$\begin{cases} \max & M(nf) \\ \max & \pi(nf) \\ \text{s.t.} & d_i(z) \geq d_i^{\min} \ \forall i \\ & \alpha \in [\alpha^{\min}, \alpha^{\max}] \\ & z \in FR \subset \mathbb{R}^2 \end{cases} \tag{4.1}$$

where the parameters $d_i^{\min} > 0$ and $\alpha^{\min} > 0$ are given thresholds, which guarantee that the new facility is not located over a demand point and that it has a minimum level of quality, respectively. The parameter $\alpha^{\max}$ is the maximum value that the quality of a facility may take in practice. By FR we denote the region of the plane where the new facility can be located.

Notice that (4.1) is a particular case of (1.4), in which $f_1(y) = -M(nf)$, $f_2(y) = -\pi(nf)$ and the feasible set S is given by the constraints in (4.1).

4.2 A new method for approximating the Pareto-front

FEMOEA is an evolutionary algorithm devised to cope with nonlinear multi-objective problems. Its main objective is to provide a good fixed size approximation of the Pareto-front, i.e., a fixed number of well-distributed and non-dominated solutions. However, it has been designed to do it *quickly*. To this aim, it combines ideas from typical algorithms described in literature for solving general multi-objective optimization problems: an *external archive* is used to store *preferable* non-dominated solutions [104, 121] (see Subsection 4.2.1), and the *crowded comparison operator* is used to guide the algorithm towards a uniformly spread Pareto-front approximation [40]. Additionally, it also inherits some concepts from other evolutionary algorithms devised to cope with single-objective optimization problems. More precisely, FEMOEA shares some ideas with UEGO. In particular, species are used, and the concept of a decreasing radius, as a mechanism of maintaining a balance between exploration and exploitation of the search space, is also considered here. Nevertheless, FEMOEA incorporates new mechanisms which help to improve the quality (efficiency) of the solutions and accelerate the optimization process. The 'improving method' or the termination criteria are two of those specific contributions.

In FEMOEA, each species is intended to occupy an efficient solution. For this purpose, FEMOEA directs the species during the searching process towards the most suitable regions. Therefore, notice that a particular species is not a fixed part of the search domain, but it can move through the space as the search proceeds. 'Species-management' is one of the core parts of FEMOEA. It consists of procedures for creating and selecting species during the whole optimization process. Additionally, FEMOEA includes an improving method, which has been logically separated from the species-management. This means that FEMOEA can be easily adapted to solve any other multi-objective problem, only adapting the improving technique.

4.2.1 Main concepts in FEMOEA

Species: center and radius

The most important concept in FEMOEA is that of species. As in UEGO, a species is defined by a center and a radius. The center is a solution and the radius is a positive

number which determines the subregion of the search space covered by that species. The main aim of the radius is to focus the searching operators on the corresponding subregions. It is worth mentioning that the radius of a species is neither constant along the execution of FEMOEA nor the same for each species. On the contrary, the radius is a monotonous function that decreases as the optimization process moves forward. Then, at each stage of the algorithm, several species with different radii can coexist simultaneously. The use of different radii throughout the optimization process allows, on the one hand, to identify regions in the search space with high quality solutions and, on the other hand, not to waste too much time in regions of the search space which are either already explored or do not provide high quality solutions [15]. This idea of a decreasing radius is a legacy of UEGO [125].

Species attributes in the objective space

Apart from the center and the radius, a species has two attributes which are related to the objective space: *the non-domination rank* (d_{rank}) and the *crowding distance* (c_{dist}), see [40]. The non-domination rank indicates the number of species which dominate that particular species. In this sense, a zero value means that such a species is not dominated by any of the remaining ones in the current population. On the other hand, the crowding distance is an estimation of the density of solutions surrounding a particular solution in a population. In this thesis, it is computed as the Euclidean distance between the two closest solutions on either side of the point in the (normalized) criterion space. An algorithm which calculates the crowding distance of each point in a population POP was proposed in [40], but using the rectangular distance instead of the Euclidean distance. That algorithm has been simplified to the case where only two objectives are considered and modified to work with the Euclidean distance, see Algorithm 9. In this work, Euclidean distance has been used since it represents the crowding better than the rectangular distance.

In Algorithm 9, f_l^i (with $l = 1, 2$) refers to the l-th objective function value of the sp_i-th point in the set POP, and $f_l^{(\max)}$ and $f_l^{(\min)}$ refer to the maximum and minimum objective function values of the l-th objective function, respectively.

Algorithm 9: Crowding distance assignment(P)

1: $n_{POP} = |POP|$
2: POP=sort(POP) {Sort using the first objective function value}
3: $c_{dist}^{(1)} = c_{dist}^{(n_{POP})} = \infty$ {In this way, boundary points are always selected}
4: **for** $sp_i = 2$ to $n_{POP} - 1$ **do**
5: $c_{dist}^{(i)} = \sqrt{\left(\frac{f_1^{(i-1)} - f_1^{(i+1)}}{f_1^{(\max)} - f_1^{(\min)}}\right)^2 + \left(\frac{f_2^{(i-1)} - f_2^{(i+1)}}{f_2^{(\max)} - f_2^{(\min)}}\right)^2}$

Lists of species

During the optimization process, two lists of species are kept by FEMOEA, whose maximum size $L_{\max}$, the same for both lists, is a given input parameter. The parameter $L_{\max}$ refers to the desired number of points in the final set approximating the Pareto-front. The first list, named *population_list*, is composed of $L_{\max}$ diverse species with different attributes, i.e. various radii, non-domination ranks and crowding distances. FEMOEA is in fact a method for managing this list (i.e. creating, deleting and improving species). The second list, called *external_list*, can be understood as a deposit to keep non-dominated solutions. Notice that the number of non-dominated points may be fewer than $L_{\max}$ during the early stages of the optimization algorithm and hence, the *external_list* may contain fewer elements than the desired ones. In fact, it cannot be guaranteed that $L_{\max}$ non-dominated solutions have been found once the termination criteria have been satisfied, although this has always been the case in our computational experiments. When this is not the case, the *external_list* and the *population_list* are joined and the $L_{\max}$ elements with the most *preferable* solutions is offered as solution (see Definition 9).

Crowed comparison operator

Definition 9. *A solution sp_i is* preferable *to a solution $sp_{i'}$, $sp_i \succ sp_{i'}$, if*

- $d_{rank}^i < d_{rank}^{i'}$, *or*
- $d_{rank}^i = d_{rank}^{i'}$ *and* $c_{dist}^i > c_{dist}^{i'}$.

The previous relation is known as *crowded comparison operator* (see [40]). To accelerate the selection process, both lists are always sorted according to the crowded

Algorithm 10: Algorithm FEMOEA

```
1: Init_species_lists
2: while termination criteria are not satisfied do
3:    Create_new_species(evals)
4:    if (length(population_list) > L_max) then
5:       Select_species(population_list)
6:    Improve_species(population_list)
7:    Update_external_list
8:    if length(external_list) > L_max then
9:       Select_species(external_list)
10:   Improve_species(external_list)
11: if length(external_list) < L_max then
12:   Compose_pareto
```

comparison operator, i.e. in ascending order according to non-domination rank, and in descending order of the crowding distance when several elements share the same non-domination rank.

4.2.2 The FEMOEA algorithm

A global description of FEMOEA is given in Algorithm 10. In the following, the different key stages in the algorithm are described:

- *Init_species_lists:* In this procedure, as many species as parameter $L_{\max}$ indicates are created. The centers of the species are randomly computed, while the radii will be the ones associated at level 1. Since such a radius coincides with the diameter of the search space, the whole search area will be covered. The *population_list* is initialized from this set of species, while the *external_list* consists only of the non-dominated species.

After this procedure, the FEMOEA main loop starts, which basically consists of three procedures: creating, improving and selecting species. This loop is executed until a stopping condition is fulfilled, namely, whenever a considerable improvement of the Pareto-front (placed in *external_list*) is not obtained among three consecutive approximations or a number of maximum levels is achieved. The number of levels (cycles or generations) will be given by the input parameter $t_{\max}$.

- *Create_new_species(evals):* For every species in the *population_list*, $evals/2$ random trial points in the area defined by its radius are created. $evals$ refers to the budget of function evaluations available for each existing species for creating a new offspring. In the computational studies $evals = 20$ has been set.

  Furthermore, for each new random candidate solution, the *closest point* (in the objective space) in the *external_list* is calculated. Then, a new random point is computed in the segment joining the candidate solution with its closest point. Notice that the intermediate point can be placed outside the area covered by the original species. If the intermediate point dominates the candidate solution, then it will be included in the *population_list* as a new species. On the contrary, if the candidate solution is the one which dominates the other, it will be the one inserted in the population. Additionally, if the two points are *indeterminate* (not one dominates the other), then they will both be inserted as new species. The radius assigned to each new species is the one associated with the current level t. The radius of a species created at level t, R_t, is given by a decreasing exponential function, where $R_{t_{\max}}$ and R_1 are the given (input parameters) smallest and largest radii. For a detailed description of how to compute the radius at each level of the algorithm, see [125]. It is interesting to remark that a location in the search space can belong to different species with different radii. Therefore, species with small radii examine a relatively small area, their motion in the space is slower, but they are able to differentiate between efficient solutions which are very close. On the contrary, species with large radii study a somewhat bigger region, they may move greater distances and discover new promising areas, which may be analyzed conscientiously in later stages of the algorithm.

  Additionally, both the non-domination rank and the crowding distance associated to each new species are computed. The *population_list* is then sorted according to the crowding comparison operator.

- *Select_species(list):* If *list* reaches its maximum allowable capacity, a decision has to be made to determine which individuals should be kept and not removed. The selection strategy used in this work is based on the *crowded comparison operator* [40]. Then, the most preferable species will be selected, i.e. between two species with different non-domination rank, the one with the lower rank is preferred.

Otherwise, the one which is located in a region with the fewest number of points (i.e., the highest crowding distance) is chosen.

It is worth mentioning here that the selection procedure could be based on other measures, such as the *density estimation criterion* of SPEA2 [162]. This technique was also studied with a comprehensive set of test problems. However, the obtained results, in terms of the distribution of the points in the Pareto-front (spread), were similar for both methods.

- *Improve_species(list):* Most classical multi-objective optimization algorithms use improving methods based on mutation operators for pushing a solution towards the optimal Pareto-front. In those methods, only the objective function values are used to guide the search strategy [40, 121, 162]. Such algorithms start from a solution. Thereafter, based on a pre-specified transition rule, the algorithm suggests a search direction. A mutation is then performed along the search direction to find a better solution. If a better solution is found, it becomes the new solution and the above procedure is continued repeatedly a number of times. Those improving methods are usually slow, requiring many function evaluations for convergence, although they can be applied to many problems without a major change in the algorithm.

  For bi-objective problems, a new gradient-based improving method has been designed, which will be discussed in Subsection 4.2.3. Broadly speaking, gradient-based methods use the first-order derivatives of the objective functions to guide the search process, which helps to quickly converge to near-optimal solutions.

  Improve_species applies the improving method to all the species in the *list*. As can be observed in Algorithm 10, this technique is applied to both the *population_list* and the *external_list* at different stages of the optimization process, i.e., steps 6 and 10, respectively. Depending on the input list, there are minor changes in the improving method (see Step 17 of Algorithm 11).

  Once all the species in the input *list* have invoked the improving method, they are reordered according to the crowded comparison operator.

- *Update_external_list*: During the previous process, new non-dominated points may be generated. In Step 7 of Algorithm 10, the *external_list* is updated by

copying the non-dominated solutions of the *population_list* to it. Of course, this implies that the points in the *external_list* dominated by the new ones have to be removed, and a reordering of the remaining ones according to the new values of the crowded comparison operator has to be performed.

- *Compose_pareto*: The solution provided by the algorithm must include $L_{\max}$ solutions since it is a requirement imposed by the user. If the number of solutions in the *external_list* reaches this value, the Pareto-set presented as the final solution will be the one kept on that list. Notice that, in the *external_list*, the non-dominated solutions which are better spread during the optimization process have been stored. However, it may happen that the number of non-dominated solutions found by the algorithm is smaller than $L_{\max}$. In such a case, a joint list will be composed considering all the elements in *population_list* and *external_list*, and the $L_{\max}$ most preferable solutions among them will be offered as a result by the algorithm.

4.2.3 The improving method

In this work, a new method to improve the efficiency of points in nonlinear bi-objective optimization problems is introduced. Basically, the algorithm looks for a search direction based on gradient information. Then, a local optimizer is applied along the suggested search direction to improve the current solution. Hybridization of multi-objective evolutionary algorithms with local search algorithms has been investigated for more than one decade. However, the use in particular of gradient-based information in this area is still scarce. In some cases, the original problem is converted into a single-objective problem to which a single-objective local search is applied [95]. In other cases, a single-objective local search is repeated to (some of) the objective functions [19]. More recently, gradient information has been used to obtain directions which simultaneously improve all the objective functions [17, 18, 64].

In this thesis, this last strategy has been followed. To find improving directions, several attempts are performed. They will be explained next. Furthermore, to move along the improving direction, a multi-objective variant of the local procedure SASS [147], has been developed. In fact, and unlike the other algorithms in literature, it does not move only along the improving direction, but *in a neigborhood of it*. In this way,

a better spread can be obtained. Finally, the main steps of the improving method will be depicted.

Looking for an improving direction

The strategies introduced next closely follow the ones described in [63]. Consider the general problem (1.4), defined in Subsection 1.1.2, but only with two objective functions, f_1 and f_2. Let $y \in S$ be a feasible point of (1.4). The method wants to find an *improving direction*, i.e., a vector $v \neq 0, v \in \mathbb{R}^n$, such that

$$f_1(y + \delta v) < f_1(y) \text{ and } f_2(y + \delta v) < f_2(y)$$

for a small value $\delta > 0$.

Checking the coordinate directions

This study checks whether the coordinate directions satisfy the previous condition. Let $\nabla f_l = (\nabla_1 f_l, \ldots, \nabla_n f_l)$ be the gradient of the objective function f_l, $l = 1, 2$.

- If $\nabla_j f_1(y) < 0$ and $\nabla_j f_2(y) < 0$ for some j (i.e., both objectives are decreasing along the j-th variable), then the vector $v = (0, \ldots, 1^{(j}, \ldots, 0)$ is an improving direction.
- If $\nabla_j f_1(y) > 0$ and $\nabla_j f_2(y) > 0$ for some j (i.e., both objectives are increasing along the j-th variable), then the vector $v = (, 0, \ldots, -1^{(j}, \ldots, 0)$ is an improving direction.
- Otherwise, there is no coordinate direction which is an improving direction.

Looking for interior directions

If none of the coordinate directions is an improving direction, we can still try to find an improving direction as follows.

Assume that one of the objective functions is monotonous along a given coordinate direction, say j, and the other objective function is monotonous along a different coordinate direction, say j', $j' \neq j$. In this case, we can still try to find out whether both objective functions are decreasing along a given direction v different from the coordinate directions. In particular, the method will study it for directions which are

linear combinations of the j-th and j'-th coordinate directions, i.e., for directions v of the form $v = (v_1, \ldots, v_n)$, with $v_k = 0\ \forall k \neq j, j'$.

Remember that the monotonicity of the objective function $f_l, l = 1, 2$, along a direction v is given by the directional derivative of f_l along the vector v, $D_v f_l$, and if the differential function of f_l at y is denoted by $\mathrm{d}f_l(y)$, then

$$D_v f_l(y) = \lim_{h \downarrow 0} \frac{f_l(y + hv) - f_l(y)}{h} = \mathrm{d}f_l(y)(v) = \nabla f_l(y) \cdot v.$$

In particular, if v is as described above, then

$$D_v f_l(y) = v_j \nabla_j f_l(y) + v_{j'} \nabla_{j'} f_l(y).$$

Notice that the method is looking for a direction v such that

$$D_v f_l(y) < 0, l = 1, 2. \tag{4.2}$$

Theorem 10. *Let $f_1, f_2 : \mathbb{R}^n \to \mathbb{R}$ be two real functions, $y \in S$ a feasible point. Suppose that there is no coordinate direction along which both functions are either increasing or decreasing. Then*

C.1. If $\nabla_j f_1(y) > 0, \nabla_{j'} f_2(y) > 0$ and $\dfrac{\nabla_j f_2(y)}{\nabla_{j'} f_2(y)} - \dfrac{\nabla_j f_1(y)}{\nabla_{j'} f_1(y)} > 0$, then any vector of the form $v = (0, \ldots, -1^{(j}, \ldots, v_{j'}, \ldots, 0)$, with

$$v_{j'} \in \left(\frac{\nabla_j f_1(y)}{\nabla_{j'} f_1(y)}, \frac{\nabla_j f_2(y)}{\nabla_{j'} f_2(y)} \right)$$

is an improving direction.

C.2. If $\nabla_j f_1(y) < 0, \nabla_{j'} f_2(y) > 0$ and $\dfrac{\nabla_j f_1(y)}{\nabla_{j'} f_1(y)} - \dfrac{\nabla_j f_2(y)}{\nabla_{j'} f_2(y)} > 0$, then any vector of the form $v = (0, \ldots, 1^{(j}, \ldots, v_{j'}, \ldots, 0)$, with

$$v_{j'} \in \left(-\frac{\nabla_j f_1(y)}{\nabla_{j'} f_1(y)}, -\frac{\nabla_j f_2(y)}{\nabla_{j'} f_2(y)} \right)$$

is an improving direction.

C.3. If $\nabla_j f_1(y) > 0, \nabla_{j'} f_2(y) < 0$ *and* $\dfrac{\nabla_j f_1(y)}{\nabla_{j'} f_1(y)} - \dfrac{\nabla_j f_2(y)}{\nabla_{j'} f_2(y)} > 0$, *then any vector of the form* $v = (0, \ldots, -1^{(j}, \ldots, v_{j'}, \ldots, 0)$, *with*

$$v_{j'} \in \left(\frac{\nabla_j f_2(y)}{\nabla_{j'} f_2(y)}, \frac{\nabla_j f_1(y)}{\nabla_{j'} f_1(y)} \right)$$

is an improving direction.

C.4. If $\nabla_j f_1(y) < 0, \nabla_{j'} f_2(y) < 0$ *and* $\dfrac{\nabla_j f_2(y)}{\nabla_{j'} f_2(y)} - \dfrac{\nabla_j f_1(y)}{\nabla_{j'} f_1(y)} > 0$, *then any vector of the form* $v = (0, \ldots, 1^{(j}, \ldots, v_{j'}, \ldots, 0)$, *with*

$$v_j \in \left(-\frac{\nabla_j f_2(y)}{\nabla_{j'} f_2(y)}, -\frac{\nabla_j f_1(y)}{\nabla_{j'} f_1(y)} \right)$$

is an improving direction.

Proof. C1. Since f_1 is increasing along the j-th coordinate direction and f_2 along the j'-th, the improving method will study if there exists a vector v of the form $v = (v_1, \ldots, v_n)$, with $v_j, v_{j'} < 0$ (it can be assumed, without loss of generality, that $v_j = -1$) and $v_k = 0 \ \forall k \neq j, j'$, such that condition (4.2) holds.

This happens if there exists a $v_{j'} \in \mathbb{R}^-$ such that

$$-\nabla_j f_l(y) + v_{j'} \cdot \nabla_{j'} f_l(y) < 0, \quad l = 1, 2,$$

that is, if

$$v_{j'} \cdot \nabla_{j'} f_l(y) < \nabla_j f_l(y), \quad l = 1, 2. \tag{4.3}$$

Since $\nabla_{j'} f_1(y) < 0$ (otherwise both functions would be increasing along the j'-th coordinate direction), for $l = 1$ condition (4.3) becomes

$$v_{j'} > \frac{\nabla_j f_1(y)}{\nabla_{j'} f_1(y)}$$

whereas the condition for $l = 2$ is

$$v_{j'} < \frac{\nabla_j f_2(y)}{\nabla_{j'} f_2(y)}.$$

Thus, in this case, both objective functions decrease along direction v whenever

$$\frac{\nabla_j f_1(y)}{\nabla_{j'} f_1(y)} < v_{j'} < \frac{\nabla_j f_2(y)}{\nabla_{j'} f_2(y)}$$

(notice that both fractions are negative, thus $v_{j'}$ is negative too), or in other words, whenever

$$\frac{\nabla_j f_2(y)}{\nabla_{j'} f_2(y)} - \frac{\nabla_j f_1(y)}{\nabla_{j'} f_1(y)} > 0.$$

C2. Since f_1 is decreasing along the j-th coordinate direction and f_2 is increasing along the j'-th, and the method wants a direction v along which both objective functions are decreasing, it will research if there exists a vector v, with $v_j > 0$, $v_{j'} < 0$ (it can be assumed, without loss of generality, that $v_j = 1$) and $v_k = 0\ \forall k \neq j, j'$, such that condition (4.2) holds. Since $v_j = 1$ and $v_{j'} < 0$, condition (4.2) holds if

$$\nabla_j f_l(y) + v_{j'} \cdot \nabla_{j'} f_l(y) < 0, \quad l = 1, 2,$$

that is, if

$$v_{j'} \cdot \nabla_{j'} f_l(y) < -\nabla_j f_l(y), \quad l = 1, 2. \tag{4.4}$$

As in C1, $\nabla_{j'} f_1(y) < 0$, thus, for $l = 1$ condition (4.4) becomes

$$v_{j'} > -\frac{\nabla_j f_1(y)}{\nabla_{j'} f_1(y)}$$

whereas the condition for $l = 2$ is

$$v_{j'} < -\frac{\nabla_j f_2(y)}{\nabla_{j'} f_2(y)}.$$

Thus, both objective functions decrease along direction v whenever

$$-\frac{\nabla_j f_1(y)}{\nabla_{j'} f_1(y)} < v_{j'} < -\frac{\nabla_j f_2(y)}{\nabla_{j'} f_2(y)},$$

that is, whenever

$$\frac{\nabla_j f_1(y)}{\nabla_{j'} f_1(y)} - \frac{\nabla_j f_2(y)}{\nabla_{j'} f_2(y)} > 0.$$

C3. Now f_1 is increasing along the j-th coordinate direction and f_2 is decreasing along the j'-th, so it will be studied if there exists a vector v, with $v_j < 0$, $v_{j'} > 0$ (it can be assumed, without loss of generality, that $v_j = -1$) and $v_k = 0\ \forall k \neq j, j'$, such that condition (4.2) holds. In this case condition (4.2) holds if

$$-\nabla_j f_l(y) + v_{j'} \cdot \nabla_{j'} f_l(y) < 0, \quad l = 1, 2,$$

that is, if

$$v_{j'} \cdot \nabla_{j'} f_l(y) < \nabla_j f_l(y), \quad l = 1, 2. \tag{4.5}$$

Since $\nabla_{j'} f_1(y) > 0$ (otherwise both functions would be decreasing along the j'-th coordinate direction), for $l = 1$ condition (4.5) becomes

$$v_{j'} < \frac{\nabla_j f_1(y)}{\nabla_{j'} f_1(y)}$$

whereas the condition for $l = 2$ is

$$v_{j'} > \frac{\nabla_j f_2(y)}{\nabla_{j'} f_2(y)}.$$

Thus, both objective functions decrease along direction v whenever

$$\frac{\nabla_j f_2(y)}{\nabla_{j'} f_2(y)} < v_{j'} < \frac{\nabla_j f_1(y)}{\nabla_{j'} f_1(y)},$$

that is, whenever

$$\frac{\nabla_j f_1(y)}{\nabla_{j'} f_1(y)} - \frac{\nabla_j f_2(y)}{\nabla_{j'} f_2(y)} > 0.$$

C4. Since f_1 is decreasing along the j-th coordinate direction and f_2 along the j'-th, the method will observe if there exists a vector v, with $v_j, v_{j'} > 0$ (it can be assumed, without loss of generality, that $v_j = 1$) and the rest of the components equal to zero, such that condition (4.2) holds. And it holds if

$$\nabla_j f_l(y) + v_{j'} \cdot \nabla_{j'} f_l(y) < 0, \quad l = 1, 2,$$

that is, if

$$v_{j'} \cdot \nabla_{j'} f_l(y) < -\nabla_j f_l(y), \quad l = 1, 2. \tag{4.6}$$

Since $\nabla_{j'} f_1(y) > 0$ (as in case C3), for $l = 1$ condition (4.6) becomes

$$v_{j'} < -\frac{\nabla_j f_1(y)}{\nabla_{j'} f_1(y)}$$

whereas the condition for $l = 2$ is

$$v_{j'} > -\frac{\nabla_j f_2(y)}{\nabla_{j'} f_2(y)}.$$

Thus, both objective functions decrease along direction v whenever

$$-\frac{\nabla_j f_2(y)}{\nabla_{j'} f_2(y)} < v_{j'} < -\frac{\nabla_j f_1(y)}{\nabla_{j'} f_1(y)},$$

that is, whenever

$$\frac{\nabla_j f_2(y)}{\nabla_{j'} f_2(y)} - \frac{\nabla_j f_1(y)}{\nabla_{j'} f_1(y)} > 0.$$

□

A good choice for $v_{j'}$ could be to set it equal to the midpoint of the mentioned intervals.

Moving the point along the improving direction

In this work, a modified version of the local optimizer SASS has been developed. The SASS algorithm was initially proposed by Solis and Wets in [147] to cope with single-objective optimization problems. It has been successfully applied to many optimization problems, see for instance [110, 133]. Here, it has been adapted, on the one hand, to work on multi-objective optimization problems and, on the other hand, to move the input point along a given improving direction, instead of randomly looking for one. The proposed algorithm will be called MO_SASS throughout this chapter. The way the heuristic MO_SASS works is described in Algorithm 11.

The algorithm MO_SASS can be applied to an arbitrary multi-objective optimization problem over a bounded subset of $\mathbb{R}^n$, although internally it assumes, as the original

SASS does, that the range in which each variable is allowed to vary is the interval $[0, 1]$. The new points are generated using a Gaussian perturbation $\xi \in \mathbb{R}^n$ over the search point y, a normalized bias term $b \in \mathbb{R}^n$ and an improving direction v to direct the search. In this way, given y, a first trial point, $y + \xi \cdot v$ is considered, and if it dominates y, then $y + \xi \cdot v$ updates the initial point, but maintaining the same radius value. Otherwise, if y and $y + \xi \cdot v$ are indeterminate solutions, then $y + \xi \cdot v$ is compared pairwise to the points in the *external_ list*. If it is dominated by any point from such a list, it is discarded; otherwise, it is stored in the *external_ list*. Notice that, as a consequence of this inclusion, there may be dominated solutions in the *external_ list*. In such a case, those solutions are removed.

The coefficient values 0.4, 0.2 and 0.5 in steps 22, 24 and 26, used for updating the bias term b are retained from Solis and Wets's results [147]. The standard deviation σ specifies the size of the sphere that most likely contains the perturbation vector, whereas the bias term b locates the center of the sphere based on directions of past successes. The size of the standard deviation of the normalized perturbation ξ_{aux} is controlled by the repeated number of successes, $scnt$, or failures, $fcnt$. A success occurs when the new point dominates the initial one. The contraction (ct) and expansion (ex) constants, as well as the upper bound σ_{ub} on the standard deviation σ, are set by the user.

As mentioned earlier, depending on the stage a species is created, this species has an associated radius that determines a subregion of the search space covered by that species. This means that any single step taken by the improving method in a given species is no longer than the radius of the species.

Since in MO_SASS the standard deviation σ specifies the size of the sphere that most likely contains the normalized perturbation vector, its upper bound σ_{ub} should have the same value than the normalized radius of the caller species. That is why the parameter σ_{ub} is also considered an input argument of MO_SASS. For the same reason, the improving vector v must also be normalized.

It is worth mentioning that the use of MO_SASS allows, on the one hand, to push y towards the true Pareto-set (steps 14-15 in Algorithm 11) and, on the other hand, to study its surrounding area to obtain indeterminate solutions (steps 17-18 in Algorithm 11). The inclusion of indeterminate points in the *external_ list* may improve the quality of the final Pareto-front, but it increases the computational effort (the more elements on the list, the more computing time required to order it). Notice that

Algorithm 11: Algorithm MO_SASS(y, σ_{ub}, v, bel)

1: Set $iter = 1$, $y^{(iter)} = y$, $b^{(iter)} = 0$, $scnt = 0$, $fcnt = 0$, $\sigma^{(0)} = \sigma_{ub}$, $\sigma_{lb} = \max\{\sigma_{ub}/1000, 10^{-5}\}$
2: Fix ex, ct, $Scnt$, $Fcnt$, $Maxfcnt$, $iter_{max}$
3: **while** $iter < iter_{max}$ and $fcnt < Maxfcnt$ **do**
4: $\sigma^{(iter)} = \sigma^{(iter-1)}$
5: **if** $scnt > Scnt$ **then**
6: $\sigma^{(iter)} = ex \cdot \sigma^{(iter-1)}$
7: **if** $fcnt > Fcnt$ **then**
8: $\sigma^{(iter)} = ct \cdot \sigma^{(iter-1)}$
9: **if** $\sigma^{(iter)} < \sigma_{lb}$ **then**
10: $\sigma^{(iter)} = \sigma_{ub}$ and $b^{(iter)} = 0$
11: **if** $\sigma^{(iter)} > \sigma_{ub}$ **then**
12: $\sigma^{(iter)} = \sigma_{ub}$
13: Generate a multivariate Gaussian random vector $\xi_{aux}^{(iter)} = N(b^{(iter)}, \sigma^{(iter)} I)$
14: **if** $y^{(iter)} + \xi^{(iter)} v$ dominates $y^{(iter)}$ **then**
15: $y^{(iter+1)} = y^{(iter)} + \xi^{(iter)} v$; $scnt = scnt + 1, fcnt = 0$
16: **else**
17: **if** $bel = 0$ and $y^{(iter)} + \xi^{(iter)} v$ is not dominated by any point in the *external_ list* **then**
18: Include $y^{(iter)} + \xi^{(iter)} v$ in *external_ list*; $scnt = 0, fcnt = 1$
19: **if** $fcnt = 0$ **then**
20: **for** $j = 1..3$ **do**
21: **if** $v_j > 0$ **then**
22: $b_j^{(iter+1)} = 0.4\xi_{aux_j}^{(iter)} + 0.2b_j^{(iter)}$
23: **else**
24: $b_j^{(iter+1)} = b_j^{(iter)} - 0.4\xi_{aux_j}^{(iter)}$
25: **else**
26: $b^{(iter+1)} = 0.5b^{(iter)}$, $fcnt = fcnt + 1$, $scnt = 0$
27: $iter = iter + 1$
28: Return $y^{(iter)}$

MO_SASS is called (through the *Improving_ method*) by FEMOEA to improve both the *population_ list* and the *external_ list*, which may mean a large number of indeterminate points. In order to reach a compromise between quality in the final Pareto-front and computational effort, it was considered the decision of not executing steps 17-18 when the *external_ list* is considered input in the *Improving_ method*. The input parameter *bel* tells MO_SASS whether the solution y belongs to the *population_ list* ($bel = 0$)

or the *external_ list* ($bel = 1$).

The stopping rules are determined by the maximum number of iterations ($iter_{\max}$) and by the maximum number of consecutive failures ($Maxfcnt$). After a comprehensive computational study, they have been set to 400 and 20, respectively.

Structure of the improving method

The main steps of the improving method are enumerated in Algorithm 12. As can be observed, initially, the coordinate directions are analyzed to discover possible improving directions. Then, if none of the coordinate directions is an improving direction ($ic = 0$), the interior directions are studied.

It is interesting to remark that all the coordinate directions are examined, in such a way that the MO_SASS algorithm can be invoked n times at most with different improving vectors as input. However, for the interior directions, only the first case which fulfills the conditions is considered.

Algorithm 12: Improve_species(*list*)

```
1: for each species y in list do
2:    ic = 0
3:    Compute ∇f_l = (∇_1 f_l, ..., ∇_n f_l), l = 1, 2
4:    for each coordinate j do {(Check for coordinate directions)}
5:       if coordinate direction j is an improving direction then
6:          Compose the improving direction vector v
7:          y = MO_SASS(y, σ_ub, v, bel).
8:          ic = ic + 1.
9:    if ic = 0 then {(Check for interior directions)}
10:      for a = 1..4 do
11:         if condition C.a of Theorem 10 is satisfied then
12:            Compose the improving direction vector v accordingly
13:            y = MO_SASS(y, σ_ub, v, bel)
14:            Break
15: Return list
```

4.2.4 FEMOEA termination criteria

Usually, the termination criteria defined by the multi-objective optimization algorithms described in literature are based on the number of function evaluations [40, 121, 162]. So, an algorithm stops when it reaches a maximum number of evaluations. However, for the problem at hand, another stopping rule based on the well-known Hausdorff distance is proposed. Informally, it measures how far two sets are from each other. Mathematically, the modified Hausdorff distance hd used in FEMOEA is given by:

$$hd(Q_1, Q_2) = \frac{\frac{\sum_{a \in Q_1} min\{d(a,b): b \in Q_2\}}{max\{d(a,a'): a,a' \in Q_1\}} + \frac{\sum_{b \in Q_2} min\{d(a,b): a \in Q_1\}}{max\{d(b,b'): b,b' \in Q_2\}}}{2},$$

where Q_1 and Q_2 are two given discrete sets and $d(\cdot,\cdot)$ is a distance function (the Euclidean distance has been used).

Specifically, the algorithm finishes if during three consecutive iterations, the changes experimented in the *external_list* are negligible (in terms of the objective function values), for a given tolerance *tol* (for this work, $tol = 10^{-7}$), i.e. the algorithm stops at iteration t provided

$$hd(f(external\_list^{(t)}), f(external\_list^{(t-1)})) < tol, \text{ and}$$
$$hd(f(external\_list^{(t-1)}), f(external\_list^{(t-2)})) < tol.$$

Nevertheless, as a safeguard, a second termination criterion based on the number of iterations executed by FEMOEA has been considered. Then, the algorithm stops if the previous condition holds *or* a maximum number of iterations has been fulfilled. This maximum value is represented by the input parameter $t_{\max}$.

Notice that the first stopping criterion allows the algorithm to stop when a good approximation of the Pareto-front is obtained, without reaching the maximum number of iterations. This allows the algorithm the saving of a lot of CPU time in some instances.

4.2.5 FEMOEA input parameters

Five input parameters must be provided by the user:

- $L_{\max}$: The number of solutions which must compose the final approximation of

the Pareto-front.

- $t_{\max}$: The maximum number of levels (or iterations).
- R_1 and $R_{t_{\max}}$: The radius that is associated with the minimum and maximum level, respectively.
- *tol*: The tolerance associated with the termination criterion.

Notice that the only parameters which really need to be fine tuned are $R_{t_{\max}}$ and $t_{\max}$. The remaining ones are either a determination of the user based on his/her experience, requirements or needs (as occurs with the value of $L_{\max}$ and *tol*), or a parameter associated with the particular problem to be handled (R_1).

4.2.6 Computational studies

All the computational studies have been run in the supercomputer Arabi of the Supercomputing Center of Murcia, Spain. In these studies, each problem was run in one of the cores of the nodes with 16GB memory (one problem at a time). As for the interval B&B method (iB&B), the implementation by B. Tóth introduced in [63] has been employed, which uses the interval arithmetic of the PROFIL/BIAS library [105] and the automatic differentiation of the C++ Toolbox library [85]. FEMOEA has been implemented in C++.

In order to have an overall view of the performance of the algorithm, different types of problems have been generated, varying the number $i_{\max}$ of demand points, the number $j_{\max}$ of existing facilities and the number k of those facilities belonging to the chain. The settings used were ($i_{\max} = 25, 50, j_{\max} = 2, k = 1$), ($i_{\max} = 25, 50, j_{\max} = 5, k = 1, 2$) and ($i_{\max} = 25, 50, j_{\max} = 10, k = 2, 4$). For every setting, 10 instances were generated by randomly choosing the parameters of the problems uniformly within pre-defined intervals (see Appendix). The searching space proposed in [135] has also been considered here for every problem.

As a general rule, the algorithm iB&B has been executed considering a tolerance of $eps = 0.03$ (the maximum width of a box on the solution list), which is not a negligible value. Even so, the algorithm ran out of memory when trying to solve several instances. In each of those cases, the value of *eps* was progressively increased until

the algorithm was able to solve that particular problem. Notice that the *eps* value is related to the quality of the final solution (the larger the value of *eps*, the greater the intervals containing the exact Pareto set). Regarding FEMOEA, it was found that a good parameter setting to deal with the current multi-objective optimization problem is: $t_{\max} = 30$, $R_{t_{\max}} = 0.005$ and $tol = 10^{-7}$. The parameter R_1 coincides with the diameter of search space. Furthermore, FEMOEA has been analyzed for two different values of the number of points in the Pareto-front, $L_{\max} = 200$ or 400.

To measure the performance of FEMOEA, two aspects are under consideration, that of effectiveness and that of efficiency [89]. Notice that for stochastic algorithms, performance indicator values are also stochastic. That is why for each random indicator, the expected value is approximated by taking the average over 5 runs.

As an effectiveness metric, whether the heuristic algorithm has successfully found an approximation of the Pareto-front was checked. A success means that both the objective function values of the solution points are included in the corresponding intervals provided by the iB&B method, and the points themselves are included in the corresponding solution boxes offered by iB&B. This is the first aim of the study, that is, to prove that all the solution points provided by FEMOEA are certainly efficient points (or are very close to efficient points). Additionally, for measuring the goodness of an approximation to the whole Pareto front, the so-called *hypervolume* measure, *hyper*, has also been computed [160]. Notice that the *hyper* value becomes larger as the number of points in the Pareto-front increases. In this sense, track of the number of generated points $L_{\max}$ in the set approximating the Pareto-front should be kept, since theoretically speaking, higher values of *hyper* represent better approximations of the Pareto-front. For the iB&B algorithm, the interval $[lowH, uppH]$, whose lower limit gives the hypervolume obtained from the upper-right corner of the boxes in the solution list of iB&B and whose upper limit gives the hypervolume obtained from the lower-left corners, is provided. It is important to mention that, prior to the computation of that hypervolume, dominated points are removed. $[lowH, uppH]$ contains the exact hypervolume of the true Pareto-front. Furthermore, since the number of boxes in the solution list provided by iB&B is usually very high (with a mean value of 37017 and a standard deviation of 36287) such an interval may be considered a good estimation of that value. The second aim of the study is to prove that if FEMOEA approximates the Pareto-front with enough points, then the corresponding hypervolume lies in the

$Time_{iBB}$	eps	$Time_{200}$	$Time_{400}$	$[lowH, uppH]$	$hyper_{200}$	$hyper_{400}$
209	0.03	262	537	[146.317,146.532]	146.320	146.332
489197	0.05	347	1144	[1.326,1.328]	1.323	1.326
508028	0.03	273	596	[3.157,3.159]	3.146	3.158
97404	0.03	310	730	[112.931,113.380]	112.728	112.956
551537	0.05	308	910	[1.751,1.764]	1.754	1.759
4536	0.03	274	562	[553.147,554.340]	553.151	553.163
569547	0.03	309	756	[2.282,2.283]	2.278	2.282
81687	0.03	266	596	[428.519,429.229]	427.964	428.800
389738	0.04	338	1072	[1.342,1.344]	1.340	1.343
281464	0.04	339	989	[1.762,1.763]	1.758	1.761
297340	0.05	302.6	789.2	[125.253,125.512]	125.176	125.288

Table 4.1: Hypervolume and computing time for problems with setting $(25, 5, 2)$.

interval.

It is worth mentioning that FEMOEA approximates the Pareto front with 100% success for all the problems (for both $L_{\max} = 200$ and $L_{\max} = 400$), i.e. its solutions are always included in the intervals provided by the iB&B algorithm. Therefore, only hypervolume results will be shown as effectiveness measurement (see the last two columns of tables 4.1 and 4.2, where the hypervolume when $L_{\max} = 200$ and $L_{\max} = 400$ points are used to approximate the Pareto-front, are shown, respectively).

As for the efficiency of the algorithm, one should measure the effort made to obtain the final result. Here, the average computing time in the five runs for FEMOEA to reach the result ($Time_{200}$ when $L_{\max} = 200$ and $Time_{400}$ when $L_{\max} = 400$) has been measured. For the shake of completeness, the computing time for iB&B ($Time_{iBB}$) is also provided.

Table 4.1 shows the computing time and the hypervolume metric obtained by both iB&B and FEMOEA (for both $L_{\max} = 200$ and $L_{\max} = 400$), for a set of 10 instances with setting $(25, 5, 2)$. Additionally, the tolerance required by iB&B to solve a particular problem is also shown (see column *eps*). The average values for the 10 problems have been computed and they are shown in the last line of the table. Notice that, instead of computing the average value of *eps* in the last column, the maximum value of *eps* in the ten instances is reported. As can be observed, the iB&B algorithm is very erratic regarding computing time, while the evolutionary algorithm seems to be more

(i_{max}, j_{max}, k)	$Time_{iBB}$	$max(eps)$	$Ni_{0.03}$	$Time_{200}$	$Time_{400}$	$[lowH, uppH]$	$hyper_{200}$	$hyper_{400}$
(25,2,1)	304440	0.04	5	210	670	[199.982,200.466]	199.679	200.065
(25,5,1)	235880	0.05	7	248	706	[75.006,76.363]	75.018	75.087
(25,5,2)	297340	0.05	6	302	789	[125.253,125.512]	125.176	125.288
(25,10,2)	161270	0.07	7	441	962	[422.501,423.243]	421.768	422.615
(25,10,4)	149140	0.05	8	446	980	[354.444,355.200]	353.694	354.478
(50,2,1)	91113	0.04	8	368	935	[155.721,156.230]	155.659	155.860
(50,5,1)	51111	0.04	8	555	1245	[105.892,106.135]	105.822	105.982
(50,5,2)	92467	0.04	6	528	1156	[113.764,114.256]	113.690	113.858
(50,10,2)	244080	0.04	7	870	1807	[107.110,107.459]	106.976	107.207
(50,10,4)	123380	0.07	9	905	1838	[78.797,79.164]	78.834	78.903
All	175022	0.07	7.1	487	1109	[173.8470,174.4028]	173.6316	173.9343

Table 4.2: Average values for Hypervolume and computing time for problems.

regular, i.e. it always spends similar computing time for instances with the same setting. Additionally, it is difficult to determine the suitable tolerance to execute iB&B for a given problem in advance. As can be seen on this table, there exist four cases where a different value of $eps = 0.03$ had to be considered.

FEMOEA increases its *hyper* value as the number of points $L_{\max}$ in the Pareto-front increases. As can be observed on Table 4.1, the hypervolume covered by the heuristic with 400 points is always included in the interval $[lowH, uppH]$. On the other hand, with 200 points in the Pareto-front, the hypervolume is smaller than the lower limit for 7 out of 10 problems. This clearly shows that in order to have a good approximation of the Pareto-front for this difficult nonlinear multi-objective problem 200 points are not enough.

For the sake of brevity, the particular results for the rest of the settings are not shown; but in order to have a general overview, the summarizing values of the last lines (as in Table 4.1) are detailed. Table 4.2 encapsulates those results. Additionally, a new column has been included ($Ni_{0.03}$), which indicates the number of instances executed by iB&B with a tolerance value of $eps = 0.03$. The remaining instances have been run with larger values of eps, being $max(eps)$ the maximum considered value. Again, a summarizing last line has been included.

As can be observed from the results with $L_{\max} = 400$, the FEMOEA average hypervolume is always included in the corresponding iB&B intervals, which means that the Pareto-fronts provided by FEMOEA cover in practice all the area of the true Pareto-front. It is important to mention that the hypervolume obtained by the heuristic with

$L_{\max} = 400$ points is always included in the interval $[lowH, uppH]$ for any particular instance. Of course, depending on the particular problem, fewer points may be required. See, for example, the value of $hyper_{200}$ for the settings $(25, 5, 1)$ and $(50, 10, 4)$, where $L_{\max} = 200$ points are enough to, in average, cover the true Pareto-front.

Additionally, iB&B was able to solve 71% of the test problems with $eps = 0.03$ (see Table 4.2). The average computing time spent by the exact algorithm for solving this subset of instances was 118080 seconds (32.8 hours). On the other hand, the subset of problems executed by iB&B with higher values of *eps* involves an average execution time of 284340 seconds (78.9 hours). This means that, independent of the complexity of the instance at hand and the *eps* tolerance required to handle it, the iB&B method has been completely superseded by the evolutionary algorithm in terms of computing time: FEMOEA needs less than 1% of the computing time of iB&B, and this considering $L_{\max} = 400$ points in the Pareto-front. Remember, however, that iB&B is an exact method which computes a set of boxes guaranteed to contain the complete efficient set, whereas FEMOEA, despite its good effectiveness in results (hypervolume), provides just an approximation of it.

4.3 High performance computing

As can be seen in previous computational studies, when the set approximating the Pareto-front must have many points (because a high precision is required), then the computational time needed by FEMOEA may not be negligible at all. Furthermore, the computational resources needed may be so high that a PC may run out of memory. In those cases, parallelizing the algorithm and running it in a supercomputer may be the best way forward. As far as the author's knowledge is concerned, the development of parallel multi-objective evolutionary algorithms is a booming field, which has not been explored enough (see [4]).

In this thesis, a parallel algorithm with application to the bi-objective facility location problem described in Section 4.1, called FEMOEA-Paral, is presented in Subsection 4.3.1. Then, its efficiency and effectiveness is analyzed in Subsection 4.3.2.

4.3.1 FEMOEA-Paral

The programming paradigm used to parallelize FEMOEA may be considered a coarse-grain model. Similar to previous proposals, each processor executes FEMOEA independent of the remaining ones most of time but considering a smaller *population_list*. More precisely, the length of such a list will be equal to $L'_{\max} = L_{\max}/P$, assuming that P processors will be available. This list will be named *local_population_list* in the following. Therefore, the idea is that different processors work with a smaller and different species list in such a way that, when merging all the local lists, a population list similar to that of the sequential version can be obtained. Notice that the species in the population list can create a new offspring or be improved without participation of the remaining ones. Consequently, there exists an intrinsic parallelism which will consist of dividing the species among the number of processors.

Nevertheless, although there exits no relationship among species in the population list, the evolution of the population list highly depends on the solutions stored in the external list. Furthermore, the external list may be modified (increasing, removing or updating species) by procedures initially applied over the population list. Then, to prevent poor effectiveness, the *external_list* is not divided among the processors, on the contrary, each processor has a local copy of it. Such a list will be called *local_external_list* throughout this thesis. An important issue to highlight is that, unlike the sequential version, those two lists are not sorted by the crowded comparison operator, but only by the first objective function value. Since the selection will be carried out in parallel, the maintenance of sorted lists by crowded comparison operator is counter-productive in terms of efficiency.

Apart from these two lists, another list, called *auxiliary_external_list* is maintained during the optimization process. Such a list is only stored at the processor with identification number 0 and keeps the most preferable species found during the whole optimization process.

Algorithm 13 sketches the structure of the parallel algorithm. In the following, the different key stages are described.

- *Init_species_lists_paral*: Initially, $L'_{\max}$ species are created at each processor (Step 1 of Algorithm 13). As in the sequential version, the center of the species are randomly computed, while the radii will be the one associated at level 1.

Algorithm 13: FEMOEA-Paral

```
1: Init_species_lists_paral
2: while termination criteria are not satisfied do
3:    Create_new_species_paral(evals)
4:    Select_species_paral(local_population_list)
5:    Improve_species_lists_paral
6:    Update_local_external_lists
7:    Select_species_paral(local_external_list)
8:    Improve_external_list_paral
9: if length(auxiliary_external_list) < L_max then
10:   Compose_pareto
```

The *local_ population_ list* is initialized from this set of species, while the *local_ external_ list* will consist only of the non-dominated species.

After this procedure, a loop starts, which basically creates, selects and improves species. This loop is executed until a stopping condition is fulfilled, namely, whenever a considerable improvement is not obtained among three consecutive approximations of the local Pareto-front (placed in *auxiliary_ external_ list*) or the maximum level $t_{\max}$ is achieved. Notice that the termination criteria is controlled by processor P_0. Then, if the stopping rule is satisfied, it sends a flag to the remaining processors indicating that the optimization procedure has finished.

- *Create_ new_ species_ paral(evals):* Each processor executes the sequential *Create_ new_ species* procedure but considering its own *local_ population_ list*, with length equal to $L'_{\max} = L_{\max}/P$ (see Subsection 4.2.2). It is important to mention that, after the creation procedure, the length of the *local_ population_ list* at each processor is usually much larger than $L'_{\max}$.

- *Select_ species_ paral (list):* In addition to the previous procedure, during the optimization process, the *local_ population_ list* as well as the *local_ external_ list* may be modified by including new species or modifying the existing ones. Therefore, following the spirit of the sequential version, a selection procedure is required. In order to prevent a decrement in the effectiveness, selections are not carried out locally at each processor. Instead, a global selection considering all the lists at the P processors is accomplished. This may imply that large amounts

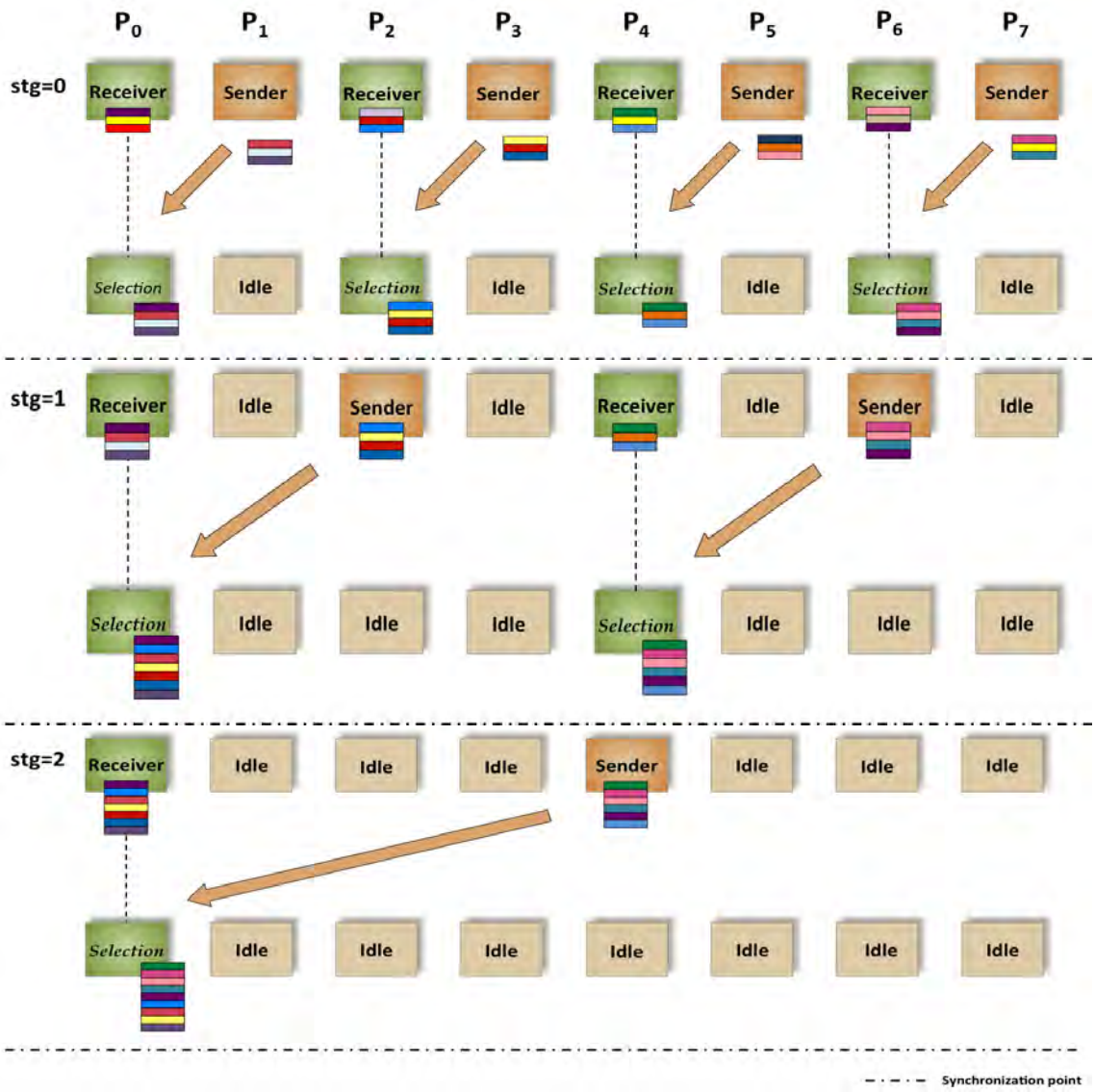

Figure 4.1: Select_species_paral procedure.

of data must be frequently transferred among processors. To speed-up the communication overheads and hence the selection procedure, a hierarchical tree communication schema has been designed. Figure 4.1 represents the global idea of this communication model. Let us assume that the root of the tree is processor P_0. Then, processor P_0 may be understood as the collector of all the transferred information.

The maximum width of the hierarchical tree is given by the number of available processors P, which are identified by P_{id} with $id \in [0, P-1]$. Its maximum number of stages (or levels) is given by $stg^{\max} = \log_2(P)$. Each stage has associated a figure $stg \in [0, stg^{\max}]$. There exist three kinds of processors: senders, receivers and idle processors. As can be observed in Figure 4.1, the role of each processor varies through the communication model.

When a processor is a receiver, it obtains a list from a sender processor, composes a joint list considering the own species list and the received one and computes the d_{rank} value associated to each species. Then, a selection is carried out, which will vary depending on the transferred *list* and the stage of the communication model.

- If *list* refers to *local_ population_ list*. In this case, if $stg < stg^{\max}$, the selection is only carried out in terms of domination ranks. More precisely, the minimum non-domination rank $d_{rank}^{\min}$ in such a way that *at least* $2 \cdot L'_{\max} \cdot 2^{stg+1}$ species exists with a non-domination rank smaller than or equal to $d_{rank}^{\min}$ is computed. Those species with $d_{rank} \geq d_{rank}^{\min}$ will be removed from the joint list. The remaining ones will be transferred through the communication hierarchical tree. In the last stage of the selection procedure, i.e. $stg = stg^{\max}$, then the receiver processor (P_0) is the hierarchical tree root, and it will select the $L_{\max}$ most preferable species by using the crowded comparison operator described in Subsection 4.2.1. The resulting *population_ list* will then be distributed back among all the processors, in such a way that $L'_{\max}$ species will be sent directly to each processor. Figure 4.2 depicts the global idea of this communication procedure.
- If *list* refers to *local_ external_ list* and $stg < stg^{\max}$, only the non-dominated species will be maintained on the joined list and transferred to the next stage of the hierarchical communication tree. If $stg = stg^{\max}$, processor P_0 (the receiver) will reduce the joint list to the $L_{\max}$ most preferable species and will store it in its own *local_ external_ list*. Then, P_0 distributes its *local_ external_ list* directly among all the processors as explained before (see Figure 4.2).

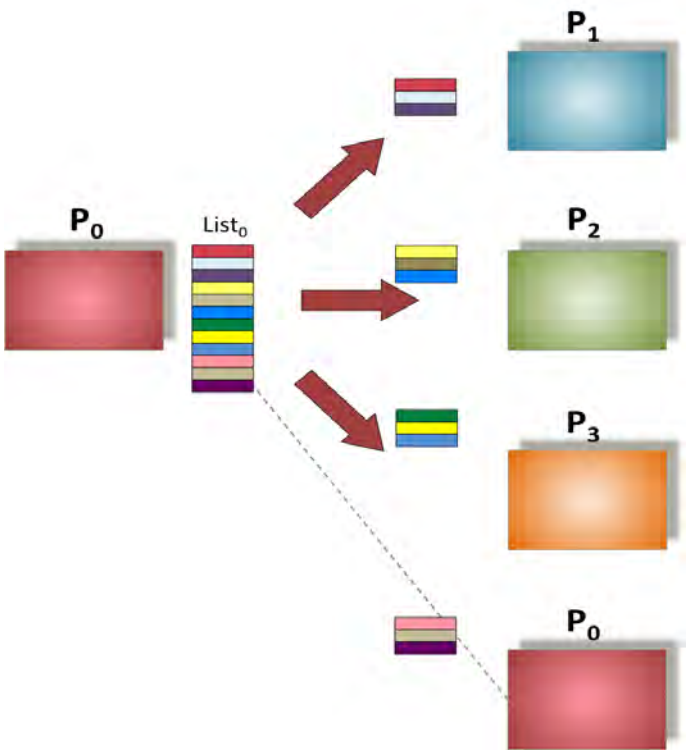

Figure 4.2: Distribution of a list carried out by processor P_0.

- *Improve_ species_ lists_ paral:* This procedure executes the *Improve_ species(*list*)* algorithm described in Subsection 4.2.3, where the input list is the *local_ population_ list*, which has a length equal to $L'_{\max}$. It is important to remember here that, as a consequence of this method, the species belonging to the *local_ population_ list* may be improved, i.e. substituted by species which dominate them. Additionally, this procedure may also add new points to the *local_ external_ list* when indeterminate species are found (see Subsection 4.2.3).

- *Update_ local_ external_ lists*: After the previous procedure, the local population list may contain species which deserve to be included in the external list. Then, similar to the sequential version, the *local_ external_ list* is updated by copying the non-dominated species of the *local_ population_ list* to it. Of course, this implies that the points in the *local_ external_ list* dominated by the new ones have to be removed.

  Finally, a selection procedure is carried out over the *local_ external_ list* where

the $L_{\max}$ most preferable solutions are chosen.

- *Improve_external_list_paral:* Each processor applies the *Improve_species(*list*)* over the received species (from the *local_external_list* sent by P_0) and sends them back to processor P_0 by using the communication model depicted in Figure 4.1. If $stg < stg^{\max}$, the selection procedure carried out by the receiver processors will maintain the non-domination species. On the contrary, if $stg = stg^{\max}$, processor P_0 will join the *auxiliary_external_list* to the received improved species, and will apply a selection procedure, where only the $L_{\max}$ most preferable species are maintained. Finally, P_0 will directly send the whole *auxiliary_external_list* to each processor (see Figure 4.2), which becomes their new *local_external_list*

- *Compose_pareto*: As has been stated, the solution provided by the algorithm must include $L_{\max}$ species. If the number of species in the *auxiliary_external_list* reaches this value, the set offered as an approximation of the Pareto-set will be the one kept on that list. Otherwise, the local population lists are sent to P_0 using the hierarchical tree communication schema explained before (see Figure 4.1), which joins these lists to the *auxiliary_external_list*, and the $L_{\max}$ most preferable species will be offered as a result by the algorithm.

4.3.2 Computational studies

As in the previous chapters, the performance of the parallel algorithm has been studied in terms of effectiveness and efficiency. To this aim, a single problem per configuration (of those solved in Subsection 4.2.6) has been run with different number of processors $P = 1, 2, 4, 8, 16, 32, 64$. However, in this case, the behavior of FEMOEA-Paral has been analyzed with higher values of $L_{\max}$. In particular, each problem has been solved with $L_{\max} = 400, 800, 1600$. Again, to counteract the randomness effect, each problem has been executed 5 times and average values have been computed.

The effectiveness has been tested, on the one hand, by checking if all the solution points provided by FEMOEA-Paral are certainly efficient points (or are very close to efficient points). To this aim, similar to the sequential version, whether both the objective function values of the solution points are included in the corresponding intervals provided by the iB&B method, and the points themselves are included in the corre-

sponding solution boxes offered by iB&B is studied. If so, a success can be considered. On the other hand, whether the corresponding hypervolume *hyper* lies in the interval $[lowH, uppH]$ provided by iB&B as well as the behavior of both FEMOEA and FEMOEA-Paral when the number of points in Pareto-front, $L_{\max}$, increases has been tested.

It is worth mentioning that FEMOEA-Paral has approximated the Pareto front with 100% success for all the executed problems. Furthermore, the obtained *hyper* values have always been included on the interval $[lowH, uppH]$ for any particular instance. Finally, as expected, that the *hyper* value increases as the number of points $L_{\max}$ in the Pareto-front does has been observed. On the other hand, that the hypervolume obtained by the parallel version is similar to the one obtained by the sequential method has been checked. This has been a challenge, since the sequential version has global control over both the population and the external lists. On the contrary, the parallel version has carried out some decisions based on local knowledge. The designed selection procedures have helped to counteract these drawbacks.

To measure the computational effort of obtaining the solutions, the efficiency *Eff* measure has been computed. Table 4.3 summarizes the average results for different values of P and grouped them according to the value of $L_{\max}$. The average *hyper* has also been included. As can be seen, the higher the $L_{\max}$ values, the larger the *hyper*.

Concerning the efficiency, it increases up to 8 processors, and then it starts to decrease. Even so, promising results have been obtained and values very superior to the ideal case have been achieved in most cases.

It is important to mention that the tendencies observed in Table 4.3, are not a

	$L_{\max}$=400			$L_{\max}$=800			$L_{\max}$=1600		
P	*hyper*	*Time*	*Eff*	*hyper*	*Time*	*Eff*	*hyper*	*Time*	*Eff*
1	93.1227	1439	-	93.1988	8186	-	93.2359	36028	-
2	93.1273	280	2.57	93.2003	1473	2.78	93.2365	8736	2.06
4	93.1271	128	2.82	93.2005	318	6.43	93.2365	1718	5.24
8	93.1266	72	2.50	93.2005	167	6.11	93.2365	729	6.18
16	93.1268	49	1.82	93.2003	110	4.64	93.2364	295	7.63
32	93.1270	43	1.06	93.2001	90	2.84	93.2362	231	4.87
64	93.1258	38	0.59	93.2001	84	1.53	93.2362	218	2.58

Table 4.3: Average results. Average iB&B's Hypervolume $[lowH, uppH] = [93.08106, 93.24334]$.

P	Creation	Selection (P_{pop} lists)	Improving (P_{pop} lists)	Selection (ext.lists)			Improving (ext.lists)			$Time$	Eff
				T_{co}	T_{cm}	Total	T_{co}	T_{cm}	Total		
1	128	713	12171	6157	-	6157	680	-	680	21537	-
2	64	601	2225	1104	954	2057	349	31	380	5774	1.86
4	32	119	388	117	193	310	192	35	227	1105	4.87
8	20	84	171	33	106	139	103	35	138	567	4.75
16	8	72	67	7	75	82	43	32	75	314	4.29
32	4	69	32	3	83	87	22	32	54	256	2.63
64	2	66	16	2	96	98	11	33	44	238	1.41

Table 4.4: Times employed by FEMOEA-Paral in some steps of the algorithm for the problem (50,10,4).

consequence of computing average values. On the contrary, the behavior is similar when particular instances are analyzed.

Taking the spectacular efficiency results into account, a deeper study is demanded. To this aim, a particular problem (the one with setting $(50, 10, 4)$) has been selected and analyzed. Table 4.4 shows the computing time employed by the steps of both FEMOEA and FEMOEA-Paral which can influence the efficiency, see algorithms 10 and 13, respectively.

Column 'Creation' refers to *Create_new_species* procedure when $P = 1$ and to *Create_new_species_paral* for larger values of P. Notice that, ideally, the *Create_new_species_paral* method should reduce the computational time associated to the sequential creation procedure proportionally to the number of processors P, since the length of *local_population_list* is reduced to $L'_{\max} = L_{\max}/P$. As can be seen in the table, the prospects are almost satisfied. It is important to mention that the smaller the length of the population list, the smaller the number of new species inserted in it.

Column 'Selection (P_{pop} lists)' involves the sequential *Select_species(population_list)* procedure and the *Select_species_paral(local_population_list)* method. The objective of both methods is to obtain a population with the $L_{\max}$ most preferable solutions. Notice that, in order to maintain the effectiveness of the sequential version and compensate its global knowledge of the population, at each stage of the hierarchical communication tree, the intermediate selections carried out by the parallel method were relaxed by allowing redundant work, i.e. $2 \cdot L'_{\max} \cdot 2^{stg+1}$ were maintained at each $stg < stg^{\max}$. The key point is that the parallel procedure works with smaller list lengths, which permits to reduce the computing time of the sequential version, although not proportionally to the number of processors. Ideally, the smaller the list

lengths, the better the efficiency. However, due to the fact that when the number of processors increases, the list size at each processor decreases, it turns out that the smaller the list sizes, the higher the communication overheads. This explain this tendency in the efficiency, i.e. it increases for up to 8 processors and decreases from this value of P.

Column 'Improving (P_{pop} lists)' refers to the *Improve_ species(population_ list)* and *Improve_ species_ lists_ paral* of algorithms 10 and 13, respectively. Notice that, at the beginning of these procedures, the length of the population to be improved as well as the length of the external list is $L_{\max}$ for the sequential version. On the contrary, for the parallel version, the lengths of the species population and the external list are $L'_{\max}$ and $L_{\max}$, respectively. Additionally, remember that, as a consequence of the application of those methods, the species in the corresponding population list may be improved (updated) or new solutions may be included on the associated external list. The inclusion of a species on the external list may suppose (i) a previous comparison to determine that any point in the external list dominates that solution, (ii) a removal of solutions (on the external list) which are dominated by the new one and (iii) a sorting of the external list. Of course, the smaller the length of the population list, the fewer the number of solutions which may deserve being included in the external list and hence, the lighter the effort to accomplish the inclusion procedure. Having a look at the figures in Table 4.4, it is possible to observe that such an effort does not decreases linearly, but exponentially. This procedure is, therefore, one of the causes for those fantastic efficiency values.

The computing time associated to Step 7 in Algorithm 10 (resp. Step 6 in Algorithm 13) is not shown, because its influence on the obtained results is minimal regarding the remaining procedures.

Column 'Selection (ext.lists)' depicts the effort required to execute the *Select_ species(external_ list)* procedure in Algorithm 10 and the *Select_ species_ paral(local_ external_ list)* method in Algorithm 13. For the parallel case, such an effort has been divided between the computational cost (T_{co}) and the communication time (T_{cm}). The sequential method must select the $L_{\max}$ most preferable solutions among the ones stored in its *external_ list*, while the parallel procedure must do it from the ones kept in the P local external lists. Although the parallel method *Select_ species_ paral(local_ population_ list)* is initially similar to the *Select_ species_ paral(local_ external_ list)*, this last

procedure differs in two important issues, which may explain why the improvement (in terms of efficiency) is higher in this part of the algorithm: (i) The intermediate selections carried out through the hierarchical communication tree do not maintain redundant work and (ii) the lengths of the lists involved here are much larger than the ones in *Select_species_paral(local_population_list)*. Focusing on the computing time T_{co}, the enormous acceleration which is possible to obtain when P increases can be observed. Of course, this is because the larger the number of processors, the smaller the local external list lengths and hence, the higher the number of parallel comparisons. As can be seen in Table 4.4, the efficiency obtained by the *Select_species_paral(local_external_list)* procedure is highly reduced by the communication costs. Notice that the weight of the communication overhead is compensated by the computing cost when the number of processors is smaller than 8. From this value, the number of communications, the waiting times and the synchronization points increase, making the efficiency decrease. Even so, super efficiencies are obtained independent of the value of P. This procedure is responsible for the obtained efficiencies as well as for the tendency of decreasing from 8 processors.

Columns 'Improving (ext.lists)' shows the execution time associated with *Improve_species(external_list)* and *Improve_external_list_paral* of algorithms 10 and 13, respectively. Notice that, for the parallel version, two efforts are depicted: the computational cost (T_{co}) and the communication time (T_{cm}). As expected, the parallel version is able to reduce the computational effort of the sequential version proportionally to the number of available processors and, hence, an almost constant ideal efficiency is obtained (note that the improved external list in FEMOEA-Paral is P times smaller than the one in FEMOEA). However, such an ideal efficiency is reduced by the overheads imposed by the communications, i.e. the efficiency values decrease when the number of processors increases.

4.4 Conclusions

In this work, a new memetic bi-objective evolutionary algorithm, FEMOEA, has been proposed. In its framework, it includes a local search, which uses gradient information to improve the quality (efficiency) of the points, as well as a termination rule to stop the algorithm as soon as a good approximation of the Pareto-front is obtained.

It has been applied to a hard-to-solve competitive facility location problem. The computational studies show that all the points offered by the algorithm as an approximation of the efficient set are always included in the boxes offered by iB&B, an interval branch-and-bound algorithm able to obtain an enclosure of the true Pareto-front. Furthermore, when the number $L_{\max}$ of points approximating the Pareto-front is adequate, the hypervolume of the approximations also lies within the corresponding interval containing the hypervolume of the true Pareto-front as offered by the interval algorithm. Thus, we can conclude that all the points offered by the algorithm are (nearly) efficient and they cover all the area of the true Pareto-front.

These findings supplement another study (see [139]) where it is compared a simplified version of FEMOEA (in which the improving method does not make use of gradient information) with other widely referenced heuristic algorithms devised to cope with multi-objective problems, namely, NSGA-II [40] and SPEA2 [162]. A set of 20 benchmark problems were solved. The results in [139] show that FEMOEA is faster and provides better approximations of the efficient set, as it obtains better results for quality indicators such as the hypervolume [160], average distance [33], additive epsilon [164], spread [40] and spacing [151].

Furthermore, in this chapter, a parallel version of FEMOEA has been developed and analyzed. A comprehensive computational study has shown that FEMOEA-Paral maintains the effectiveness of the sequential version, i.e. both algorithms approximate the Pareto-front with 100% success in all the instances, their hypervolume values are always included in the interval provided by iB&B and they both obtain similar hypervolume figures for any particular instance. The maintenance of the effectiveness values has been possible thanks to the implemented selection procedure, which allows to concurrently choose the most preferable solutions. The efficiency of the parallel version has also been tested. The distribution of the computational load carried out by FEMEOA-Paral allows to highly accelerate the sequential computational times, in such a way that FEMOEA-Paral has been able to obtain super efficiency values. Additionally, the scalability of the parallel version has also been analyzed by solving instances with a larger computational burden. As expected, the bigger the instances to be solved, the better the efficiency.

Global conclusion and future work

In this thesis, new heuristic algorithms able to solve either single or multi-objective optimization algorithms have been presented. They have been applied to new facility location and design problems with endogenous demand introduced in this thesis, as well as to other location problems already proposed in literature. Furthermore, several parallel strategies have been developed in order to improve the performance of the proposed algorithms.

In order to deal with a new *single facility location problem with endogenous demand*, the evolutionary algorithm UEGO, with a new generalized Weiszfeld-like algorithm (WLMv) as local optimizer, has been considered. It has been evaluated and compared to another algorithm described in literature, i.e. an interval branch-and-bound method (iB&B). Results have shown that UEGO is able to always obtain the optimal solution, and this reducing the computational time employed by iB&B (although notice that iB&B is a rigorous method). Furthermore, for the first time in literature, an endogenous demand model has been compared to its corresponding exogenous demand model, concluding that the loss in profit due to the assumption of exogenous demand may be very high (see also [130]).

Besides, a modification of the local search WLMv, which allows to reduce UEGO's computational time even more, and its parallelization, which shows an almost linear speedup up to 8 processors, have also been proposed (see [7]).

Several algorithms have been proposed to cope with the new corresponding *leader-follower problem with endogenous demand* (or centroid problem), namely, a grid search procedure (GS), a multistart method (MSH) and an evolutionary algorithm (TLUEGO). For the last two algorithms, two versions are obtained depending on the algorithm used for solving the corresponding follower problem, i.e. UEGO and iB&B. Based on an extensive study, it has been concluded that TLUEGO, considering UEGO as follower solver, (TLUEGO_UE) is the most reliable one since it provides the best objective function value with less deviation and in less CPU time (see also [131]).

Additionally, several new creation procedures have been analyzed to increase the robustness of TLUEGO_UE at finding the global optimum. However, the computational effort needed by TLUEGO_UE with those new procedures is higher, and therefore,

high performance computing becomes a necessity. Three parallel approaches which not only allow to obtain the solution faster, but also to solve larger size instances, have been implemented. In particular, a distributed memory programming algorithm (PMP_TLUEGO), a shared memory programming algorithm (SMP_TLUEGO) and a hybrid of the two previous algorithms (HPP_TLUEGO) have been proposed. The computational studies showed that all of them have good performance behaviors for up to 8 processors (see [8]).

Finally, to deal with a *bi-objective facility location and design problem*, a new multi-objective evolutionary algorithm, called FEMOEA, has been introduced. The new evolutionary algorithm has been evaluated and compared to an exact interval B&B algorithm. Based on a comprehensive computational study, it can be concluded that all the points offered by FEMOEA are (nearly) efficient and they cover all the area of the true Pareto-front (see [132]).

Moreover, since the computational time needed by FEMOEA may be not negligible when the set approximating the Pareto-front must have many points (because a high precision is required), a parallel strategy, called FEMOEA-Paral, has been proposed. Computational studies showed that FEMOEA-Paral is able to maintain the effectiveness of the sequential version and this by highly reducing the computational costs. In fact, super efficiency values were obtained independently of the number of available processors. Furthermore, the parallel version showed its good scalability.

In the future, we plan to compare the solutions obtained in the location models when different demand generating functions $w_i(U_i)$ are used to model the variability of the demand, in order to know whether there exists a pattern in the regions of near optimality depending on the function used. Additionally, we plan to solve other location models, focusing mainly on *undesirable (obnoxious) facility location models*, which lead to hard-to-solve global optimization problems, and for which usually only ad-hoc heuristic procedures have been proposed. We also plan to develop new methods for solving some of the most successful existing models and to create new, more realistic models for the location of both undesirable and semi-desirable facilities.

Regarding FEMOEA, there are several research issues that we believe to be worth exploring in the future. One of them is when to apply the improving method. FEMOEA does it at every iteration (or level), but in other algorithms the local search is applied either only as a final step to refine the solution (see [87]) or only as a first step to

guide the evolutionary algorithm (see [91]). A common aim in both cases is to save computational time. These and other strategies should be investigated. Another issue is which individuals should the improving method be applied to. FEMOEA applies it to the $L_{\max}$ elements of the *population_list* and the *external_list*, but other strategies are also possible [18, 95]. The use of other local searches different from the improving method introduced in this work is another topic of research. Moreover, the extension of the improving method proposed in Section 4.2 to the case where more than 2 objectives are considered, is also a future line to work on.

From the high performance computing perspective, there are many aspects that have not been sufficiently analyzed. In particular, we are interested in the projection of our proposed algorithms on heterogeneous environments.

Appendix

For every setting in this thesis, the problems have been generated by randomly choosing the parameters of the problems uniformly within the following intervals:

- $locd_i, locf_j \in FR_l$,
- $Aver_{A_i}(w_i(U_i)) \in [1, 10]$,
- $\gamma_i \in [0.75, 1.25]$,
- $a_{ij} \in [0.5, 5]$,
- $G_l(nf_l) = \sum_{i=1}^{i_{\max}} \Phi_i(d_i(z_l)) + G_l^b(\alpha_l)$ where
  - $\Phi_i(d_i(z_l)) = \dfrac{Aver_{A_i}(w_i(U_i))}{(d_i(z_l))^{\phi_l^{i0}} + \phi_l^{i1}}$ with $\phi_l^{i0} = \phi^0 = 2, \phi_l^{i1} \in [0.5, 2]$,
  - $G_l^b(\alpha_l) = e^{\frac{\alpha}{\alpha_l^0} + \alpha_l^1} - e^{\alpha_l^1}$ with $\alpha_l^0 \in [5, 7], \alpha_l^1 \in [4, 4.5]$,
- $s_l \in [2, 3.5]$, the parameter for $F_l(M_l(nf_l)) = s_l \cdot M_l(nf_l)$,
- $b_1, b_2 \in [1, 2]$, parameters for $d_i(z_l) = \sqrt{b_1(x_l - locd_{i1})^2 + b_2(y_l - locd_{i2})^2}$ (see [58]),
- The functions $g_i(r)$ used to measure the attraction of a demand point towards a facility were of the the form $g_i(r) = r^2$.

Notice that in Chapter 2, a single chain is considered, $l = 1$. In Chapter 3, the leader and follower chains are considered, $l = 1, 2$. In Chapter 4, a single chain and fixed demand are considered, therefore, $l = 1$ and $Aver_{A_i}(w_i(U_i))$ is $\widehat{w}_i$.

Those intervals were obtained by varying up and down the value of the parameters of the quasi-real problem studied in [154], where a case of location of supermarkets in south-est Spain is studied. Nevertheless, when applying the model to a particular problem those parameters have to be fine-tuned.

Bibliography

[1] J.I. Agulleiro and J.J. Fernández. Fast tomographic reconstruction on multicore computers. *Bioinformatics*, 27(4):582–583, 2011.

[2] J.I. Agulleiro, E.M. Garzón, I. García, and J.J. Fernández. Vectorization with SIMD extensions speeds up reconstruction in electron tomography. *Journal of Structural Biology*, 170(3):570–575, 2010.

[3] E. Alba. *Parallel metaheuristics: a new class of algorithms*. Wiley-Interscience, 2005.

[4] E. Alba, G. Luque, and S. Nesmachnow. Parallel metaheuristics: recent advances and new trends. *International Transactions in Operational Research*, 20:1–48, 2013.

[5] AMD. Ati stream technology. `http://ati.amd.com/technology/streamcomputing/`.

[6] G. Amdahl. Validity of the single processor approach to achieving large-scale computing capabilities. In *AFIPS conference proceedings*, volume 30, pages 483–485, 1967.

[7] A.G. Arrondo, J. Fernández, J.L. Redondo, and P.M. Ortigosa. An approach for solving competitive location problems with variable demand using multicore systems. *Optimization Letters*, 2012. (doi: 10.1007/s11590-012-0596-z).

[8] A.G. Arrondo, J.L. Redondo, J. Fernández, and P.M. Ortigosa. High performance computing approaches for solving a continuous (1|1)-centroid problem with endogenous demand. Submitted to *Journal of Global Optimization*.

[9] G.L. Beane. *The effects of microprocessor architecture on speedup in distributed memory supercomputers*. Ph.D. Thesis, University of Maine, August 2004.

[10] D. Beasley, D.R. Bull, and R.R. Martin. A sequential niche technique for multimodal function optimization. *Evolutionary Computation*, 1(2):101–125, 1993.

[11] O. Berman and D. Krass. Locating multiple competitive facilities: Spatial interaction models with variable expenditures. *Annals of Operations Research*, 111(1):197–225, 2002.

[12] H. Bersini and G. Seront. In search of a good evolution-optimization crossover. In R. Männer and B. Manderick, editors, *Proceeding of PPSN-II, second international conference on parallel problem solving from nature*, pages 479–488. Elsevier, Amsterdam, The Netherlands, 1992.

[13] H.G. Beyer and H.P. Schwefel. Evolution strategies: a comprehensive introduction. *Journal Natural Computing*, 1(1):3–52, 2002.

[14] J. Bhadury, H.A. Eiselt, and J.H. Jaramillo. An alternating heuristic for medianoid and centroid problems in the plane. *Computers and Operations Research*, 30(4):553–565, 2003.

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

[16] A. Bode and M.D. Cin, editors. *Parallel computer architectures: theory, hardware, software, applications*, volume 732 of *Lecture Notes in Computer Science*. Springer, 1993.

[17] P.A.N. Bosman. On gradients and hybrid evolutionary algorithms for real-valued multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 16(1):51–69, 2012.

[18] P.A.N. Bosman and E.D. de Jong. Exploiting gradient information in numerical multi-objective evolutionary optimization. In H.G. Beyer et al. et al., editor, *2005 genetic and evolutionary computation conference (GECCO'2006)*, volume 1, pages 755–762, New York, USA, 2005. ACM Press.

[19] P.A.N. Bosman and E.D. de Jong. Combining gradient techniques for numerical multi-objective evolutionary optimization. In M. Keijzer et al. et al., editor, *2006 genetic and evolutionary computation conference (GECCO'2006)*, volume 1, pages 627–634, Seattle, Washington, USA, July 2006. ACM Press. ISBN 1-59593-186-4.

[20] M.L. Brandeau and S.S. Chiu. Location of competing facilities in a user-optimizing environment with market externalities. *Transportation Science*, 28(2):125–139, 1994.

[21] J.F. Campbell, G. Stiehr, A.T. Ernst, and M. Krishnamoorthy. Solving hub arc location problems on a cluster of workstations. *Parallel Computing*, 29(5):555–574, 2003.

[22] E. Cantú-Paz. A summary of research on parallel genetic algorithms. Technical Report IlliGAL 95007, University of Illinois at Urbana-Champaign, 1995.

[23] E. Cantú-Paz. A survey of applications and methods. Technical Report IlliGAL 97003, University of Illinois at Urbana-Champaign, 1997.

[24] E. Cantú-Paz. Designing efficient master-slave parallel genetic algorithms. In J.R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D.B. Fogel, M.H. Garzon, D.E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic programming 1998: proceedings of the third annual conference*, page 455, University of Wisconsin, Madison, Wisconsin, USA, 1998. Morgan Kaufmann.

[25] L.G. Casado and J. Fernández. Introducción a la optimización global intervalar. In *El análisis de intervalos en España: Desarrollos, herramientas y aplicaciones*, pages 45–62. Documenta Universitaria, 2005.

[26] B. Chapman, G. Jost, and R. van der Pas. *Using OpenMP. Portable shared memory parallel programming.* The MIT Press, Cambridge, Massachusetts, 2008.

[27] J.W. Chung, S.M. Oh, and C. Choi. A hybrid genetic algorithm for train sequencing in the korean railway. *Omega*, 37(3):555–565, 2009.

[28] M. Cosnard and J.L. Philippe. Achieving superlinear speedups for the multiple polynomial quadratic sieve factoring algorithm on a distributed memory multiprocessor. In *Proceedings of the joint international conference on vector and parallel processing*, pages 863–874, London, UK, 1990. Springer-Verlag.

[29] 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.

[30] T.G. Crainic, M. Toulouse, and M. Gendreau. Parallel asynchronous tabu searh for multicomodity location-allocation with balancing requirements. *Annals of Operations Research*, 63(2):277–299, 1996.

[31] D. Culler, J.P. Singh, and A. Gupta. *Parallel computer architecture: a hardware/software approach (The Morgan Kaufmann series in computer architecture and design)*. Morgan Kaufmann, 1998.

[32] T. Cura. A parallel local search approach to solving the uncapacitated warehouse location problem. *Computers and Industrial Engineering*, 59(4):1000–1009, 2010.

[33] P. Czyżżak and A. Jaszkiewicz. Pareto simulated annealing – a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7(1):34–47, 1998.

[34] V.G. da Fonseca and C.M. Fonseca. The attainment-function approach to stochastic multiobjective optimizer assessment and comparison. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Experimental methods for the analysis of optimization algorithms*, pages 103–130. Springer, 2010.

[35] P.J. Darwen and X. Yao. Speciation as automatic categorical modularization. *IEEE Transactions on Evolutionary Computation*, 1(2):100–108, 1997.

[36] C. Darwin. *The origin of species by means of natural selection*. Mentor Reprint, 1958, 1859.

[37] Y. Davidor. A naturally occurring niche and species phenomenon: The model and first results. In R.K. Belew and L.B. Booker, editors, *Proceedings of the 4th international conference on genetic algorithms*, pages 257–263. Morgan Kaufmann, 1991.

[38] A. de Bruin, G.A.P. Kindervater, and H.W.J.M. Trienekens. Towards an abstract parallel branch and bound machine. In *Solving combinatorial optimization problems in parallel - methods and techniques*, pages 145–170, London, UK, 1996. Springer-Verlag.

[39] A. de Silva and D. Abramson. A parallel interior point method and its application to facility location problems. *Computational Optimization and Applications*, 9(3):249–273, 1998.

[40] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[41] M. Dorigo. Ant colony optimization. *Scholarpedia*, 2007.

[42] M. Dorigo and G. Di 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, London, 1999.

[43] D.G. Drake. Introduction to Java threads. A quick tutorial on how to implement threads in Java. `http://www.javaworld.com/javaworld/jw-04-1996/jw-04-threads.html`.

[44] T. Drezner and Z. Drezner. Competitive location strategies for two facilities. *Regional Science and Urban Economics*, 12:485–493, 1982.

[45] T. Drezner and Z. Drezner. Facility location in anticipation of future competition. *Location Science*, 6(1):155–173, 1998.

[46] T. Drezner and Z. Drezner. Retail facility location under changing market conditions. *IMA Journal of Management Mathematics*, 13(4):283–302, 2002.

[47] T. Drezner and Z. Drezner. Finding the optimal solution to the Huff based competitive location model. *Computational Management Science*, 1(2):193–208, 2004.

[48] T. Drezner and H.A. Eiselt. Consumers in competitive location models. In Z. Drezner and H.W. Hamacher, editors, *Facility location: applications and theory*, pages 151–178. Springer, 2001.

[49] Z. Drezner, editor. *Facility location: a survey of applications and methods*. Springer, Berlin, Berlin, 1995.

[50] Z. Drezner and H.W. Hamacher. *Facility location. Applications and theory*. Springer, Berlin, 2002.

[51] Z. Drezner, G.O. Wesolowsky, and T. Drezner. On the logit approach to competitive facility location. *Journal of Regional Science*, 38(2):313–327, 1998.

[52] R. Duncan. A survey of applications and methods. *Computer*, 23(2):5–16, 1990.

[53] A.E. Eiben, P.E. Raué, and Zs. Ruttkay. Genetic algorithms with multiparent recombination. In Y. Davidor, H.P. Schwefel, and R. Männer, editors, *Parallel problem solving from nature - PPSN III*, volume 866 of *Lecture Notes in Computer Science*, pages 78–87, Berlin, 1994. Springer.

[54] A.E. Eiben and J.E. Smith. *Introduction to evolutionary computing*. Springer, 2003.

[55] H.A. Eiselt and G. Laporte. Objectives in location problems. In Z. Drezner, editor, *Facility location: a survey of applications and methods*, Springer Series in Operations Research and Financial Engineering, pages 151–180. Springer, Berlin, 1995.

[56] H.A. Eiselt and G. Laporte. Sequential location problems. *European Journal of Operational Research*, 96(2):217–231, 1996.

[57] H.A. Eiselt, G. Laporte, and J.F. Thisse. Competitive location models: a framework and bibliography. *Transportation Science*, 27(1):44–54, 1993.

[58] J. Fernández, P. Fernández, and B. Pelegrín. Estimating actual distances by norm functions: a comparison between the $l_{k,p,\theta}$-norm and the $l_{b_1,b_2,\theta}$-norm and a study about the selection of the data set. *Computers and Operations Research*, 29(6):609–623, 2002.

[59] J. Fernández and B. Pelegrín. Using interval analysis for solving planar single-facility location problems: new discarding tests. *Journal of Global Optimization*, 19(1):61–81, 2001.

[60] J. Fernández, B. Pelegrín, F. Plastria, and B. Tóth. Planar location and design of a new facility with inner and outer competition: An interval lexicographical-like solution procedure. *Networks and spatial economics*, 7(1):19–44, 2007.

[61] J. Fernández, B. Pelegrín, F. Plastria, and B. Tóth. Solving a Huff-like competitive location and design model for profit maximization in the plane. *European Journal of Operational Research*, 179(3):1274–1287, 2007.

[62] J. Fernández and B. Tóth. Obtaining an outer approximation of the efficient set of nonlinear biobjective problems. *Journal of Global Optimization*, 38(2):315–331, 2007.

[63] J. Fernández and B. Tóth. Obtaining the efficient set of nonlinear biobjective optimization problems via interval branch-and-bound method. *Computational Optimization and Applications*, 42(3):393–419, 2009.

[64] J. Fliege and B.F. Svaiter. Steepest descent methods for multicriteria optimization. *Mathematical Methods of Operations Research*, 51(3):479–494, 2000.

[65] M. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, 21(9):948–960, 1972.

[66] C.M. Fonseca, V.G. da Fonseca, and L. Paquete. Exploring the performance of stochastic multiobjective optimizers with the second-order attainment function,. In C.A. Coello, A.H. Aguirre, and E. Zitzler, editors, *Evolutionary multi-criterion optimization. Third international conference, EMO*, volume 3410 of *Lecture Notes in Computer Science*, pages 250–264. Springer, 2005.

[67] C.M. Fonseca and P.J. Fleming. Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. In S. Forrest, editor, *Proceedings of the fifth international conference on genetic algorithms*, pages 416–423, San Mateo, California, 1993. University of Illinois at Urbana-Champaign, Morgan Kauffman Publishers.

[68] S. Forrest, B. Javornik, R.E. Smith, and A.S. Perelson. Using genetic algorithms to explore pattern recognition in the immune system. *Journal of Evolutionary Computation*, 1(3):191–211, 1993.

[69] Message Passing Interface Forum. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications*, 8(3-4):165–414, 1994.

[70] R.L. Francis, T.J. Lowe, and A. Tamir. Demand point aggregation for location models. In Z. Drezner and H. Hamacher, editors, *Facility location: application and theory*, pages 207–232. Springer, 2002.

[71] R.L. Francis, L.F. McGinnis, and J.A. White. *Facility layout and location: an analytical approach*. Prentice Hall, Englewood Cliffs, 1992.

[72] F. García-López, B. Melián-Batista, J.A. Moreno-Pérez, and J.M. Moreno-Vega. The parallel variable neighborhood search for the p-median problem. *Journal of Heuristics*, 8(3):375–388, 2002.

[73] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM Parallel Virtual Machine, a user's guide and tutorial for networked parallel computing*. MIT Press, Cambridge, Massachusetts, 1994.

[74] B. Gendron and T.G. Crainic. A parallel branch-and-bound algorithm for multicommodity location with balancing requirements. *Computers and Operations Research*, 24(9):829–847, 1997.

[75] D. Ghazfan, B. Srinivasan, and M. Nolan. Massively parallel genetic algorithms. Technical Report 94-01, Department of Computer Technology. University of Melbourne, 1994.

[76] W.F. Gilreath and P.A. Laplante. *Computer architecture*. Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[77] Globus. The globus alliance. `http://www.globus.org/`.

[78] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

[79] D.E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, New York, 1989.

[80] D.E. Goldberg, K. Deb, and J.H. Clark. Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, 6:333–362, 1992.

[81] D.E. Goldberg and J. Richardson. Genetic algorithm with sharing for multimodal function optimization. In J.J. Grefenstette, editor, *Genetic algorithms and their applications*, pages 177–183. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[82] S.L. Hakimi. On locating new facilities in a competitive environment. *European Journal of Operational Research*, 12(1):29–35, 1983.

[83] S.L. Hakimi. Locations with spatial interactions: competitive locations and games. In R.L. Francis and P.B. Mirchandani, editors, *Discrete location theory*, pages 439–478. Wiley/Interscience, 1990.

[84] H.W. Hamacher and S. Nickel. Classification of location models. *Location Science*, 6(1):229–242, 1998.

[85] R. Hammer, M. Hocks, U. Kulisch, and D. Ratz. *C++ Toolbox for verified computing I: basic numerical problems: theory, algorithms and programs*. Springer-Verlag, Berlin, 1995.

[86] P. Hansen and J.F. Thisse. The generalized Weber-Rawls problem. In *Operational research '81 (Hamburg, 1981)*, pages 569–577. North-Holland, Amsterdam, 1981.

[87] K. Harada, K. Ikeda, and S. Kobayashi. Hybridizing of genetic algorithm and local search in multiobjective function optimization: recommendation of GA then LS. In M. Keijzer et al. et al., editor, *2006 genetic and evolutionary computation conference (GECCO'2006)*, volume 1, pages 667–674, Seattle, Washington, USA, July 2006. ACM Press. ISBN 1-59593-186-4.

[88] G. Harik, E. Cantu-Paz, D.E. Goldberg, and B. Miller. The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation*, 7(3):231–253, 1999.

[89] E.M.T. Hendrix and B. G. Tóth. *Introduction to nonlinear and global optimization*. Springer, New York, 2010.

[90] J.L. Hennessy and D.A. Patterson. *Computer architecture, fourth edition: A quantitative approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[91] A.G. Hernández-Díaz, C.A.C. Coello, F. Pérez, R. Caballero, J. Molina, and L.V. Santana-Quintero. Seeding the initial population of a multi-objective evolutionary algorithm using gradient-based information. In *2008 congress on evolutionary computation (CEC'2008)*, pages 1617–1624, Hong Kong, June 2008. IEEE Service Center.

[92] J.H. Holland. *Adaptation in natural and artificial systems*. The University of Michigan Press, Ann Arbor, Michigan, 1975.

[93] R. Horst and H. Tuy. *Global Optimization: Deterministic Approaches*. Springer, 3rd edition, 1996.

[94] J. Hu and E. Goodman. Robust and efficient genetic algorithms with hierarchical niching and a sustainable evolutionary computation model. In K. Deb et al., editor, *Genetic and evolutionary computation–GECCO 2004. Proceedings of the genetic and evolutionary computation conference. Part I*, volume 3102 of *Lecture Notes in Computer Science*, pages 1220–1232. Springer-Verlag, 2004.

[95] X. Hu, Z. Huang, and Z. Wang. Hybridization of the multi-objective evolutionary algorithms and the gradient-based algorithms. In *Proceedings of the 2003 congress on evolutionary computation (CEC'2003)*, volume 2, pages 870–877, Canberra, Australia, 2003. IEEE Press.

[96] D.L. Huff. Defining and estimating a trading area. *Journal of Marketing*, 28(3):34–38, 1964.

[97] T. Ibaraki. Theoretical comparisons of search strategies in branch and bound algorithms. *International Journal of Computer and Information Sciences*, 5(4):315–344, 1976.

[98] Java. Remote method invocation home. `http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp`.

[99] M. Jelásity. *The shape of evolutionary search: Discovering and representing search space structure*. Ph.D. Thesis, Leiden University, 2001.

[100] M. Jelásity, P.M. Ortigosa, and I. García. UEGO, an abstract clustering technique for multimodal global optimization. *Journal of Heuristics*, 7(3):215–233, 2001.

[101] K.A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* Ph.D. Thesis, University of Michigan, Ann Arbor, MI, 1975.

[102] R. Kicinger, T. Arciszewski, and K.A. De Jong. Evolutionary computation and structural design: a survey of the state of the art. *Computers and Structures*, 83(23-24):1943–1978, 2005.

[103] M. Kilkenny and J.F. Thisse. Economics of location: a selective survey. *Computers and Operations Research*, 26(14):1369–1394, 1999.

[104] J. Knowles and D. Corne. The Pareto archived evolution strategy: a new baseline algorithm for Pareto multiobjective optimisation. In *Proceedings of the 1999 congress on evolutionary computation*, 1999.

[105] O. Knüppel. PROFIL/BIAS - a fast interval library. *Computing*, 1(53):277–287, 1993.

[106] J.R. Koza. *Genetic programming: On the programming of computers by means of natural selection.* Cambridge, Massachusetts: MIT Press, 1992.

[107] J.R. Koza. *Genetic programming II: automatic discovery of reusable programs.* MIT Press, 1994.

[108] P.L. Lanzi, W. Stolzmann, and S.W. Wilson, editors. *Learning classifier systems, from foundations to applications*, volume 1813 of *Lecture Notes in Computer Science*. Springer, 2000.

[109] J.P. Li, M.E. Balazs, G.T. Parks, and P.J. Clarkson. A species conserving genetic algorithm for multimodal function optimization. *Evolutionary Computation*, 10(3):207–234, 2002.

[110] J.M.G. Linares, N. Guil, E.L. Zapata, P.M. Ortigosa, and I. García. Deformable shapes detection by stochastic optimization. In *2000 IEEE international conference on image processing (ICIP'2000)*, Vancouver, Canada, 2000.

[111] S. Luke. *Issues in scaling genetic programming: Breeding strategies, tree generation, and code bloat.* Ph.D. Thesis, Department of Computer Science, University of Maryland, 2000.

[112] S.W. Mahfoud. *Niching methods for genetic algorithms.* Ph.D. Thesis, University of Illinois at Urbana-Champaign, Urbana, IL, 1995.

[113] R.G. McGarvey and T.M. Cavalier. Constrained location of competitive facilities in the plane. *Computers and Operations Research*, 32:359–378, 2005.

[114] L.G. Mitten. Branch and bound methods: general formulation and properties. *Operations Research*, 18:24–34, 1970.

[115] P. Moscato. On genetic crossover operators for relative order preservation. Technical Report C3P-778, Caltech Concurrent Computation Program, 1989.

[116] P. Moscato and C. Cotta. Memetic algorithms. `http://www.lcc.uma.es/ccottap/papers/memetic_HAAM.pdf`.

[117] P. Moscato and C. Cotta. A gentle introduction to memetic algorithms. In F. Glover and G. Kochenberger, editors, *Handbook of metaheuristics*, pages 105–144. Kluwer Academic Publishers, Boston MA, 2003.

[118] P. Moscato and C. Cotta. An introduction to memetic algorithms. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial.*, 19:131–148, 2003.

[119] P. Moscato, C. Cotta, and A. Mendes. Memetic algorithms. In G.C. Onwubolu and B.V. Babu, editors, *New optimization techniques in engineering*, volume 141 of *Studies in Fuzziness and Soft Computing*, chapter 3, pages 53–86. Springer, 2004.

[120] H. Mühlenbein and H.M. Voigt. Gene pool recombination in genetic algorithms. In I.H. Osman and J.P. Kelly, editors, *Proceeding of the metaheuristics conference*, pages 53–62. Kluwer Academic Publishers, 1995.

[121] A.J. Nebro, F. Luna, E. Alba, B. Dorronsoro, J.J. Durillo, and A. Beham. AbYSS: Adapting scatter search to multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 12(4):439–457, 2008.

[122] A. Negri, D.A. Scannicchio, F. Touchard, and V. Vercesi. Multi thread programming, 2001. `https://atlserver.pv.infn.it/atlas/EventFilter/NoteMT.pdf`.

[123] A. Neumaier. Complete search in continuous global optimization and constraint satisfaction. In *Acta Numerica 2004*, pages 271–369. Cambridge University Press, 2004.

[124] P.M. Ortigosa. *Métodos estocásticos de optimización global. Procesamiento paralelo.* Ph.D. Thesis, Universidad de Málaga, 1999.

[125] P.M. Ortigosa, I. García, and M. Jelásity. Reliability and performance of UEGO, a clustering-based global optimizer. *Journal of Global Optimization*, 19(3):265–289, 2001.

[126] P.M. Ortigosa, J.L. Redondo, I. García, and J.J. Fernández. A population global optimization algorithm to solve the image alignment problem in electron crystallography. *Journal of Global Optimization*, 37(4):527–539, 2007.

[127] F. Plastria. Continuous location problems. In Z. Drezner, editor, *Facility location: a survey of applications and methods*, Springer Series in Operations Research and Financial Engineering, pages 151–180. Springer, Berlin, 1995.

[128] F. Plastria. Static competitive facility location: an overview of optimisation approaches. *European Journal of Operational Research*, 129(3):461–470, 2001.

[129] H. Ratschek and J. Rokne. *New computer methods for global optimization.* Ellis Horwood, Chichester, 1988.

[130] J.L. Redondo, J. Fernández, A.G. Arrondo, I. García, and P.M. Ortigosa. Fixed or variable demand? Does it matter when locating a facility? *Omega*, 40(1):9–20, 2012.

[131] J.L. Redondo, J. Fernández, A.G. Arrondo, I. García, and P.M. Ortigosa. A two-level evolutionary algorithm for solving the facility location and design (1|1)-centroid problem on the plane with variable demand. *Journal of Global Optimization*, To appear. (doi: 10.1007/s10898-012-9893-4).

[132] J.L. Redondo, J. Fernández, J.D. Álvarez, A.G. Arrondo, and P.M. Ortigosa. A new memetic evolutionary algorithm for approximating the Pareto-front of nonlinear bi-objective optimization problems: application to a competitive facility location problem. Submitted to *Journal of Global Optimization*.

[133] J.L. Redondo, J. Fernández, I. García, and P. M. Ortigosa. Heuristics for the facility location and design (1|1)-centroid problem on the plane. *Computational Optimization and Applications*, 45(1):111–141, 2010.

[134] J.L. Redondo, J. Fernández, I. García, and P.M. Ortigosa. Parallel algorithms for continuous competitive location problems. *Optimization Methods & Software*, 23(5):779–791, 2008.

[135] J.L. Redondo, J. Fernández, I. García, and P.M. Ortigosa. A robust and efficient global optimization algorithm for planar competitive location problems. *Annals of Operations Research*, 167(1):87–105, 2009.

[136] J.L. Redondo, J. Fernández, I. García, and P.M. Ortigosa. Solving the multiple competitive facilities location and design problem on the plane. *Evolutionary Computation*, 17(1):21–53, 2009.

[137] J.L. Redondo, J. Fernández, I. García, and P.M. Ortigosa. Parallel algorithms for continuous multifacility competitive location problems. *Journal of Global Optimization*, 50(4):557–573, 2011.

[138] J.L. Redondo, J. Fernández, I. García, and P.M. Ortigosa. Solving the facility location and design (1|1)-centroid problem via parallel algorithms. *Journal of Supercomputing*, 58(3):420–428, 2011.

[139] J.L. Redondo, J. Fernández, and P.M. Ortigosa. FEMOEA: a Fast and Efficient Multi-Objective Evolutionary Algorithm. Submitted to *European Journal of Operational Research*, 2012.

[140] J.L. Redondo, I. García, P.M. Ortigosa, B. Pelegín, and P. Fernández. Parallelization of an algorithm for finding facility locations for an entering firm under delivered pricing. In *Proceeding of parallel computing 2005 (PARCO 2005)*, pages 269–276, 2005.

[141] J.L. Redondo, I. García, P.M. Ortigosa, B. Pelegrín, and P. Fernández. Parallelization of an algorithm for finding facility locations for an entering firm under delivered pricing. In G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, and E. Zapata, editors, *Parallel computing: current and future issues of high-end computing*, volume 33 of *NIC series*, pages 269–276. John von Neumann Institute for Computing, 2006.

[142] J.L. Redondo, I. García, B. Pelegrín, P. Fernández, and P.M. Ortigosa. CG-GASUB: A parallelized algorithm for finding multiple global optima to a class of discrete location problem. In A. Paias and F. Saldanha da Gama, editors, *Proceedings of the EURO winter institute on locations and logistic*, pages 139–146. Universidade de Lisboa, 2007.

[143] C.S. ReVelle and H.A. Eiselt. Location analysis: a synthesis and survey. *European Journal of Operational Research*, 165(1):1–19, 2005.

[144] J.B. Rosen and G.-L. Xue. Computational comparison of two algorithms for the euclidean single facility location problem. *ORSA Journal on Computing*, 3(3):207–212, 1991.

[145] J. Sanders and E. Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. NVIDIA, 2010.

[146] S.G. Shiva. *Advanced computer architecture*. CRC Press, 2006.

[147] F.J. Solis and R.J.B. Wets. Minimization by random search techniques. *Mathematics of Operations Research*, 6(1):19–30, 1981.

[148] E. Speckenmeyer, B. Monien, and O. Vornberger. Superlinear speedup for parallel backtracking. In *Proceedings of the 1st international conference on supercomputing*, pages 985–993, London, UK, 1988. Springer-Verlag.

[149] S.S. Syam and M.J. Côté. A location–allocation model for service providers with application to not-for-profit health care organizations. *Omega*, 38(3-4):157–166, 2010.

[150] G. Syswerda. Simulated crossover in genetic algorithms. In L.D. Whitley, editor, *Proceeding of the second workshop on foundations of genetic algorithms*, pages 239–255. Morgan Kaufmann Publishers, 1993.

[151] K.C. Tan, C.K. Goh, Y.J. Yang, and T.H. Lee. Evolving better population distribution and exploration in evolutionary multi-objective optimization. *European Journal of Operational Research*, 171(2):463–495, 2006.

[152] A. Tórn and A. Zilinskas. *Global optimization*. Springer-Verlag New York, Inc., New York, NY, USA, 1989.

[153] B. Tóth and J. Fernández. *Interval methods for single and bi-objective optimization problems - applied to competitive facility location problems*. Lambert Academic Publishing, Saarbrücken, 2010.

[154] B. Tóth, F. Plastria, J. Fernández, and B. Pelegrín. On the impact of spatial pattern, aggregation, and model parameters in planar Huff-like competitive location and design problems. *OR Spectrum*, 31(1):601–627, 2009.

[155] E. Vallada and R. Ruiz. Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. *Omega*, 37(1-2):57–67, 2010.

[156] Official web. PVM (Parallel Virtual Machine), 2011. `http://www.csm.ornl.gov/pvm/`.

[157] E. Weiszfeld. Sur le point pour lequel la somme des distances de n points donnés est minimum. *Tohoku Mathematical Journal*, 43:355–386, 1937.

[158] L. While, L. Bradstreet, and L. Barone. A fast way of calculating exact hypervolumes. *IEEE Transactions on Evolutionary Computation*, 16(1):86–95, 2012.

[159] L. Yu, K. Liu, and K. Li. Ant colony optimization in continuous problem. *Frontiers of Mechanical Engineering in China*, 2(4):459–462, 2007.

[160] E. Zitzler. Evolutionary algorithms for multiobjective optimization: methods and applications. Master's thesis, Swiss Federal Institute of Technology (ETH) Zurich, Shaker Verlag, Germany, 1999.

[161] E. Zitzler, J. Knowles, and L. Thiele. Quality assessment of Pareto set approximations. In J. Branke, K. Deb, K. Miettinen, and R. Slowinski, editors, *Multiobjective optimization. Interactive and evolutionary approaches*, pages 373–404. Springer. Lecture Notes in Computer Science Vol. 5252, Berlin, Germany, 2008.

[162] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization. In K. C. Giannakoglou, D. T. Tsahalis, J. Périaux, K. D. Papailiou, and T. Fogarty, editors, *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*, pages 95–100, Athens, Greece, 2002. International center for numerical methods in engineering (CIMNE).

[163] E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms – a comparative study. In A. E. Eiben, editor, *Parallel problem solving from nature V*, pages 292–301, Amsterdam, September 1998. Springer-Verlag.

[164] E. Zitzler, L. Thiele, M. Laumanns, C. M. Foneseca, and V.G. da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.