High Performance Genetic Algorithm for Land Use Planning

Juan Porta, Jorge Parapar, Ramón Doallo[a], Francisco F. Rivera[b], Inés Santé, Rafael Crecente[c]

[a] {juan.porta,jparaparl,doallo}@udc.es
Computer Architecture Group. University of A Coruña, Spain
[b] ff.rivera@usc.es
D. Electronics and Computing. University of Santiago de Compostela, Spain
[c] {ines.sante,rafael.crecente}@usc.es
Land Laboratory. University of Santiago de Compostela, Spain

Abstract

This study uses genetic algorithms to formulate and develop land use plans. The restrictions to be imposed and the variables to be optimized are selected based on current local and national legal rules and experts' criteria. Other considerations can easily be incorporated in this approach. Two optimization criteria are applied: land suitability and the shape-regularity of the resulting land use patches. We consider the existing plots as the minimum units for land use allocation. As the number of affected plots can be large, the algorithm execution time is potentially high. The work thus focuses on implementing and analyzing different parallel paradigms: multi-core parallelism, cluster parallelism and the combination of both. Some tests were performed that show the suitability of genetic algorithms to land use planning problems.

Keywords: land use planning; genetic algorithms; parallel programming; distributed programming; clusters of multi-core systems; GIS

1. Introduction

Land use planning is a broad term that can be applied to different processes related to the regulation and management of land use. According to the FAO (Food and Agriculture Organization of the United Nations), land

use planning is the systematic assessment of potential land and water alternatives for land use and economic and social conditions to select and adopt the best land use options (FAO et al., 1993).

Developing a comprehensive land use plan is a long and laborious process, requiring great effort from public administrations and technical teams to achieve a good solution. As a result, there is an increasing demand for tools that support the planning process. One of the most complex tasks in this process is allocating land use categories to spatial units, resulting in a land use zoning map. Multiple conflicting objective functions and a high number of spatial units are involved in this process.

Due to these characteristics, we propose a parallel genetic algorithm for land use zoning that uses an irregular spatial structure based on a cadastral parcel map. This work is done within the frame of a project that aims to develop a spatial decision support system for the development of Municipal Land Use Plans in Galicia, an autonomous region in Northwest Spain. But as this is a general algorithm for optimizing of land use allocation, it can be used to design different types of land use plans in any region.

Our study cases are the municipalities of Galicia. This region is characterized by highly fragmented land ownership. Each municipality thus has, on average, about 36,000 plots, leading to high computational costs for executing the genetic algorithm. To the best of our knowledge, the related literature represents no solutions for handling large regions with so many plots. Therefore, one of the main issues to be considered here is using different parallelism strategies to reduce the execution time, and thus improving the algorithm results. Two levels of parallelism are considered: a lower one for multi-cores based on shared memory paradigms, and a higher one for clusters based on message passing paradigms. These two levels can be joined, giving a third hybrid one.

The algorithm has been developed using the Java programming language. In particular, implementating the multi-core parallelism uses the standard concurrent programming package included in the Java Platform, and the cluster version has been implemented using the specific Java message passing library called MPJ Express (Shafi, 2011). Java was chosen because the performance gap between it and native languages (e.g. C and Fortran) has recently narrowed, thanks to the Java Virtual Machine (JVM) Just-in-Time (JIT) compiler. Moreover, the interest for parallel programming in Java is rising because of the appealing features of this language for programming multi-core cluster architectures, particularly the built-in networking

and multi-threading support, and the continuous increase in JVM performance. Furthermore, the number of available libraries contributes to the productivity of the projects implemented in Java (Taboada, Touriño, and Doallo, 2009).

Due to the type of the information involved, using a Geographical Information System (GIS) (Nyerges and Jankowski, 2009) can be helpful. As part of this work, two user interfaces were implemented to allow users to work with these algorithms. One of these interfaces is graphical and embedded into a GIS software, which makes managing of the tool even easier (Section 7.3). The other one was intended for environments where using graphical interfaces is not allowed, e.g., clusters.

1.1. Related work

Numerous studies have recognized the multi-objective nature of land use planning problems (Janssen, van Herwijnen, Stewart, and Aerts, 2008). These objectives often include the land suitability for the land uses of a land category (Arentze, Borgers, Ma, and Timmermans, 2010; Cromley and Hanink, 1999; Eastman, Jin, Kyem, and Toledano, 1995; Xibao, Jianming, and Xiaojian, 1995) and some kind of spatial criteria, especially patch compactness (shape-regularity), where a patch is defined as a contiguous area within the same land category (Aerts, Eisinger, Heuvelink, and Stewart, 2003; Aerts, van Herwijnen, Janssen, and Stewart, 2005; Duh and Brown, 2007; Janssen, van Herwijnen, Stewart, and Aerts, 2008; Kai, Bo, Qing, and Shengxiao, 2009; Stewart, Janssen, and Herwijnen, 2004). The simplest spatial metrics for evaluating compactness are based on metrics for each land use, including the number of patches (Aerts, van Herwijnen, Janssen, and Stewart, 2005; Janssen, van Herwijnen, Stewart, and Aerts, 2008), the largest patch for each land use (Aerts, van Herwijnen, Janssen, and Stewart, 2005; Janssen, van Herwijnen, Stewart, and Aerts, 2008), or the number of neighboring cells with the same land use (Aerts, Eisinger, Heuvelink, and Stewart, 2003; Kai, Bo, Qing, and Shengxiao, 2009). More complex metrics are based on the relationship between area and perimeter for each land use patch, as the average, for all patches of the ratio of the number of perimeter cells to the total number of cells in the patch (Janssen, van Herwijnen, Stewart, and Aerts, 2008), or the average ratio of the perimeter divided by the square root of the patch area (Aerts, van Herwijnen, Janssen, and Stewart, 2005). Brookes (2001) developed a more sophisticated method for patch design based on a genetic algorithm, applicable only in a raster environment.

Spatial allocation problems involve great computational complexity, as they are combinatorial optimization problems that are characterized by a large number of alternative solutions. They thus often require high computation times (Xiao, Bennett, and Armstrong, 2001), especially when the objectives include spatial characteristics like compactness or connectedness (Duh and Brown, 2007). Due to this, and because the number of spatial units involved in a land use allocation problem is usually high, searching the optimal solution requires turning to heuristic algorithms capable of achieving near-best solutions in a reasonable time (Matthews, Sibbald, and Craw, 1999). In particular, genetic algorithms have proven efficient and effective for solving geographical problems (Xiao, Bennett, and Armstrong, 2001). Numerous studies have used genetic algorithms for spatial land use allocation (Aerts, van Herwijnen, Janssen, and Stewart, 2005; Balling, Taber, Brown, and Day, 1999; Eldrandaly, 2010; Feng and Lin, 1999; Holzkamper and Seppelt, 2007; Janssen, van Herwijnen, Stewart, and Aerts, 2008; Kai, Bo, Qing, and Shengxiao, 2009; Matthews, Sibbald, and Craw, 1999; Stewart, Janssen, and Herwijnen, 2004; Xibao, Jianming, and Xiaojian, 1995; Xin and Zhi-xia, 2008; Zhang, Zeng, and Bian, 2010). Most algorithms operate on a regular raster grid. Land use zoning based on a regular grid is unrealistic, as it may lead to a single land use plot allocated to several categories or a group of different plots allocated to a single land use category. The specific planning laws in the study area also require land use zoning based on cadastral plots. Only two of the cited studies use irregular spatial units, but the number of units is small in these cases; 155 in Balling et al. (1999) and six in Xin and Zhi-xia (2008).

The high number of plots involved in a municipal land use plan leads to high computational costs when running enough iterations to explore the complete search space. This is why we considered using parallel computing. Research on parallel computing applied to geographical information science began over 20 years ago but most current geographical analytical tools and models are still based on sequential processing (Guan and Clarke, 2010). Some researchers have started to implement GIS based on parallel environments (Huang, Wei, and Li, 2009). Armstrong and Densham (1992) describe strategies for parallelizing location-allocation models, but optimization algorithms for spatial land use allocation have not been parallelized. Finally, the architecture of processors that dominate the market presently is multi-core. This fact justifies parallelizing the proposed algorithms.

The paper is structured as follows: Section 2 introduces the concrete land

use planning problem that must be resolved. Section 3 explains the main characteristics of the genetic algorithms, formally establishes the optimization problem and discusses the representation of the individuals of the genetic problem and other data structures. Section 4 summarizes a pre-processing stage and describes the specific characteristics of the genetic algorithm. This section also explains the functions used to evaluate the land suitability and patch compactness. Section 5 introduces the implementations of the parallel versions of the algorithm. Section 6 discusses the experimental results. Section 7 provides details about other important functionalities of the proposed tool. Finally, Section 8 presents some conclusions and ideas for future work.

2. Land use planning problem definition

As reported in section 1.1, the land use planning problem can be formulated using an optimization problem in which each piece of land is allocated to the best category according to certain criteria and restrictions.

Galicia land use planning laws define a set of fifteen land use categories (Lei 2/2002, 2002; Lei 2/2010, 2010). The laws also describe the restrictions enforced for each category (i.e., the maximum and minimum size for each one) and establish conditions about when a piece of land is in one of the following circumstances: should be allocated in one category, may be allocated in one category, or may not be allocated in one category. For some categories, spatial allocation is completely and uniquely determined by legal restrictions (i.e., closeness to riversides or roads). For other categories, the conditions only partially establish their spatial allocation. We refer to these two groups of categories as fixed and non-fixed categories, respectively.

Laws and experts in land use planning issues also advise that the spatial allocation process should consider the current boundaries of the existing plots in the municipality, i.e., a plot should not be divided into several parts with different categories. A plot whose category is uniquely determined by legal conditions is called a fixed plot, and the rest are non-fixed plots.

Allocating the categories that cover urban areas and rural settlements should be considered with special attention. On the one hand, these categories generate the most social controversy among the inhabitants, mainly due to building restrictions. On the other hand, the technical conditions for their allocation are usually specific and difficult to satisfy. Therefore, allocating the rural settlements is calculated first with an evolutionary algorithm (Porta et al., 2012) and considered input for the genetic algorithm. Our main

concern in this work is to solve the problem of the remaining categories, including the urban areas to be treated like any other non-fixed category.

Computations to spatially delimit the fixed categories and determine the fixed plots can be performed easily using geometric operations (i.e., buffers, intersections, or differences). In our proposal, they are executed in a preprocessing stage. The problem is then reduced to assign one non-fixed use category to each of the non-fixed plots. If the number of non-fixed categories is C, and N is the number of non-fixed plots, the number of possible solutions is C^N. There are fewer possible solutions because of the constraints, e.g., the maximum and minimum amounts of area that can be allocated to each non-fixed category.

In general, land use planning regulations do not establish explicit methods to quantitatively evaluate land use plans to compare different alternatives. There are many variables of different natures that can be considered to validate any solution to the problem. Our study, based on experts' criteria, considers two aspects to be optimized: the compactness of the resulting land use patches and the land suitability that each plot has for the category allocated to it. These criteria can be changed in different cases accordingly.

Land use patches are defined as the polygons resulting from the union of plots allocated to the same category. Their regularity is estimated as described below, using their geometrical features (areas and perimeters). Regarding the land suitability of each plot to each category, this work considers it as a piece of input data established by experts. In (Santé-Riveira, Crecente-Maseda, and Miranda-Barrós, 2008), the authors discussed different methods for its estimation and studied methods to determine the optimal total area to be allocated to each land use. Intervals centred on these optimal values can be used to choose the upper and lower bounds of the area constraints. These bounds are also input parameters.

Hence, the objective is to maximize these optimization criteria for the search space defined by all possible combinations of non-fixed plots with non-fixed categories, subject to given area constraints.

The optimization problem can thus be established as maximizing a fitness function that represents a trade-off between the aptitude of the plots to their category and the compactness of the spots, subject to the condition that the total area of every category is inside a given interval. Section 4 provides details about this formulation and Section 3 introduces some related concepts.

3. Land use optimization using genetic algorithms

Decision makers consider several, usually conflicting, objectives in many problems like this one. They are usually called multiobjective combinatorial optimization (MOCO) problems (Ehrgott and Gandibleux, 2000). MOCO is a discrete optimization problem with several variables, several objectives and a specific constraint structure defining the feasible solution set. In our case, the categories of each N plots define the set of variables, and the objectives are the aptitude and compactness defined in section 4.2.2. Because MOCO problems have discrete natures, they are non-continuous and thus cannot be solved as linear programming problems. To solve these problems, researchers have proposed many approximation methods based on iterative heuristics that intelligently combine different concepts for exploring and exploiting the search space. One of the most robust heuristics is the genetic algorithms.

Genetic algorithms (Goldberg, 1989) are search heuristics that are often applied to optimization or learning. They are based on the principles of natural evolution and use terms as genes and individuals, and operators as selection, crossover and mutation. They use the "survival of fitness" evolutionary analogy, where the best individuals survive to the next generations. Genetic algorithms maintain a set of individuals called the population. Each individual, or chromosome, encodes a candidate solution and is composed of genes. With the *selection*, *crossover*, *mutation* and *election* operators, individuals evolve and generate a new population.

To formalize the land use optimization problem in a genetic algorithm, a way to represent individuals must be defined. In Matthews et al. (1999), the authors propose two different genotype representations. The first one is based on a one-dimensional array, the land block (LB) representation, in which each land block corresponds to a current plot. The second one is the percentage and priority (P&P) representation, in which genes hold two values, the target percentage to be allocated and the priority for each land use. The LB representation usually requires many blocks, which demands a large amount of memory. Conversely, the P&P representation often needs more operations and more iterations. There is a third method to represent the individuals based on a two-dimensional structure, called the grid method(Kai, Bo, Qing, and Shengxiao, 2009). It divides the region into rectangular cells, where each cell represents a piece of land to be classified. This method was rejected due to potential problems in matching grid-based solutions with plots. In our study, the plots usually have irregular patterns. Our proposal is to use the

first approach primarily because it fits with the plot information provided by the GIS, and the computational drawbacks can be reduced by the right data structure choice for storing the population. Parallel implementation can diminish the high execution times.

Each individual gene represents a label that identifies the possible non-fixed category allocated to the corresponding plot. Individuals consist of as many genes as the number of considered plots and can thus be large in terms of number of genes. This is why an optimal use of the available memory becomes a critical issue. As a consequence, data structures with fast access times must be carefully chosen.

To validate the quality of individuals, a fitness function with two components is considered. Section 4.2.2 explains it in-depth. Individuals that do not satisfy the restrictions are not penalized, but they are not allowed to exist in the next population.

4. Pre-processing stage and genetic algorithm implementation

This section introduces the preprocessing stage and data structures chosen to optimize data access time. Our specific problem is formulated using a genetic algorithm scheme.

4.1. Pre-processing stage and basic data structures

Some information derived from the geographical data is obtained in a pre-processing stage. This information is saved in a file to be read when the genetic algorithm starts. Both stages are thus uncoupled, and several executions of the genetic algorithm with different parameters can be easily performed without rerunning the pre-processing. Getting most information requires, executing computationally expensive geometric operations (e.g., intersections). Using the pre-processing stage, these calculations will be run just once. The time spent reading the file is negligible compared to the time spent by those operations. In the pre-processing stage, the plots with fixed category are identified and will not considered in the genetic algorithm. Furthermore, fixed plots allocated to non-fixed categories are also identified and marked in the preprocessing stage; we do not dismiss them, because they influence the compactness calculation, although they cannot change their categories.

For compactness, the lengths of the border lines between plots are needed, so each plot is checked to find all its neighbors (adjacent plots). Two plots

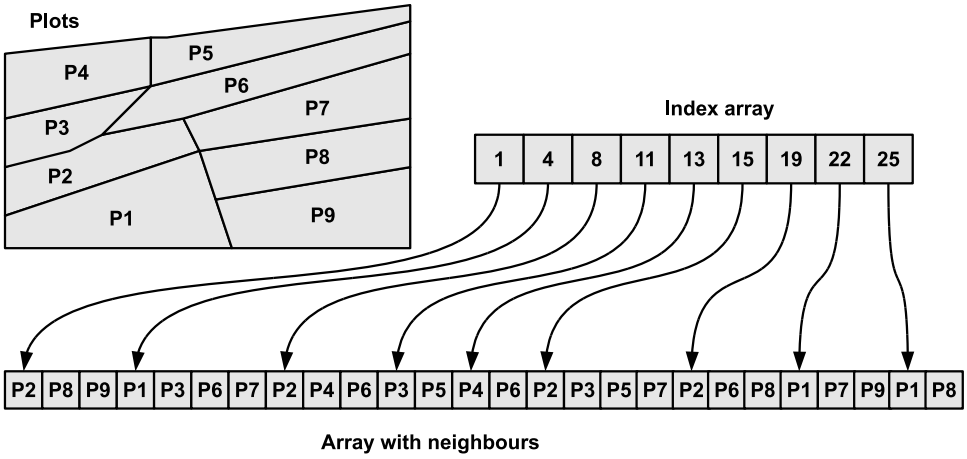

Figure 1: Arrays used to store the information about the neighborhood.

are neighbors when the length of the borders they share is greater than zero, which means that two plots are not neighbors if they only touch each other at one point. Calculating the neighbors is also a computationally expensive operation. Doing so at the pre-processing stage thus saves execution time.

Choosing the right data structure to store the neighbors and the length of its frontier is an important issue, as the genetic algorithm often accesses this information. It is thus important to minimize their access time. As the number of neighbors of each plot can differ, two unidimensional arrays are used to store the list of neighbors: an array of neighbors and an index array. The i-th entry of the index array indicates the position in which the first neighbor of the i-th plot is stored in the array of neighbors, where $i=1...N$, with N being the number of the considered plots. The neighbors of each plot are stored consecutively in the array of neighbors. Figure 1 shows an example of these two arrays. Neighbors of plot P2 are P1, P3, P6 and P7, and they are stored starting from position 4 of the array of neighbors (4 is the value of the second entry in the index array). This structure presents low latency in its access, which is much lower than a list of lists. The two-dimensional array is discarded due to the variable number of neighbors of each plot.

The pre-processing stage also involves reading other input data, including maximum and minimum areas or category weights. Some geographic information layers, like coast protection, wetlands, rivers, railway lines, roads, wind farms, or burnt areas, are also needed to determinate special zones.

As mentioned above, an individual gene is a label that represents any

category. The number of available categories is usually small, so the category can be labeled using a single byte. An individual is represented by an array of bytes with as many entries as the number of plots.

The data structure used to store the population is a two-dimensional array of bytes with one row for each individual. To store the fitness values, a one-dimensional array is also used with as many entries as the population size. To store new generations, these data structures must be duplicated, which increases the memory problem. These data could not be updated in-place, as they are needed in further steps of the genetic algorithm.

4.2. Genetic algorithm for land use planning

Consider N plots geometrically characterized by their area a_k and perimeter p_k and C categories such that each plot is assigned to only one category; ap_{ik} denotes the aptitude of plot k to the $i-th$ category established as an input value. Our problem can then be formulated as obtaining the assignment of each plot to the category that maximizes a certain objective function. This function must be defined considering both aptitude and geometrical criteria. In our proposal, the objective is to find the best assignment with the largest compacity.

Consider the set of plots assigned to each category such that N_i plots are assigned to the i-th category, and therefore a_{ij}, p_{ij} and ap_{ij} are the area, perimeter and aptitude of the j-th plot in the i-th set, respectively.

4.2.1. Genetic algorithm stages

Table 2 shows the pseudocode of our genetic algorithm. In the following paragraphs, all steps are explained using the numbered lines of the pseudocode as reference.

Let P be the population with M individuals, P_i an individual with size N (plots), $F(P_i)$ the value of the fitness function of P_i, P_{i_j} the land-use category of the j-th gene (plot) of P_i, and Q the number of genes to be changed by the mutation operator.

The initial individuals are randomly created (lines 1 to 3), but this population must be feasible in the sense that it must satisfy the restrictions imposed in the problem. For severe restrictions, not enough individuals may be created to complete the population. In that situation, the algorithm stops after several attempts to create feasible individuals. An alternative might be starting with valid solutions in the population if experts could provide some

1 for ($i = 1$ to M) {
2 randomly create P_i
3 }
4 while ($execution\_time \leq max\_time$) {
5 i = 1;
6 while ($i \leq M$) {
7 select P_{i_0} and P_{i_1} where $1 \leq i_0, i_1, \leq M$ } Selection
8 randomly select j_0 and j_1 where $1 \leq j_0 < j_1 \leq N$
9 $\tilde{P_{i_0}} = P_{i_{0_1}} ... P_{i_{0_{j_0}}} P_{i_{1_{j_0+1}}} ... P_{i_{1_{j_1}}} P_{i_{0_{j_1+1}}} ... P_{i_{0_N}}$ } Crossover
10 $\tilde{P_{i_1}} = P_{i_{1_1}} ... P_{i_{1_{j_0}}} P_{i_{0_{j_0+1}}} ... P_{i_{0_{j_1}}} P_{i_{1_{j_1+1}}} ... P_{i_{1_N}}$
11 for ($a = 1$ to Q) {
12 randomly select k_0 and k_1 where $1 \leq k_0, k_1 \leq N$
13 $\tilde{P}_{i_0 k_0} = random\_category$ } Mutation
14 $\tilde{P}_{i_1 k_1} = random\_category$
15 }
16 $P'_i = \hat{P}_\alpha$ *where* $F(\hat{P}_\alpha) =$
$= max\{F(P_{i_0}), F(P_{i_1}), F(\tilde{P_{i_0}}), F(\tilde{P_{i_1}})\}$ } Election
17 $P'_{i+1} = \hat{P}_\beta$ *where* $F(\hat{P}_\beta) =$
$= max_{2^{nd}}\{F(P_{i_0}), F(P_{i_1}), F(\tilde{P_{i_0}}), F(\tilde{P_{i_1}})\}$
18 $i = i + 2$
19 }
20 $P = P'$
21 }
22 *return* $\hat{P}_\gamma$ *where* $F(\hat{P}_\gamma) = max\{F(P_0), ..., F(P_M)\}$

Figure 2: Sequential genetic algorithm psuedocode.

such solutions. Nevertheless, in all tests with real cases, the algorithm could always generate a valid random population.

The first iteration of the main loop of the genetic algorithm is executed to obtain a new complete population. The main loop of the genetic algorithm (lines 4 to 21) is hereafter called the genetic loop, and it is the loop in which the selection, crossover, mutation and election actions are performed.

We use the roulette-wheel technique for the selection stage (Goldberg and Deb, 1991) (line 7). This is a fitness proportionate method in which individuals with higher fitness have a higher probability to be chosen. The main drawback of this method is that it could reduce the search space if

super-individuals, which are individuals with much higher fitness than the rest, are present in the population. Conversely, this method does not sort individuals according to their fitness as in rank based methods (Bäck, 1996).

A two-point crossover (Spears and DeJong, 1991) with random point-selection is implemented (lines 8 to 10). It randomly selects two different plot array positions and swaps the genes of both parents between those two positions. The user establishes the crossover probability.

The user can also set the probability of applying mutations. Theoretically, each individual gene can change with a given probability, but the number of genes to be changed (Q) can instead be determined by multiplying the given probability and individual length. The genes are randomly selected (lines 11 to 15). This strategy saves execution time. Mutation rate is an important parameter, although there is not a way to establish the best value. In fact, it is directly linked to the problem itself. The literature suggests different values for it: DeJong set it to 0.001 (Jong, 1975) and Grefenstette to 1/L (where L is the individual size) (Grefenstette, 1986). In the section devoted to the results, some conclusions about this issue are shown.

In every genetic loop iteration, two individuals are chosen to be part of the new population using elitism (lines 16 to 17). If the chosen individual already exists in the population and the user has forced the individual uniqueness as a feature of the algorithm, then it is mutated. If no valid individual is found after a number of mutations, a new one is randomly created.

The stop criterion is based on time instead of the fitness function value (line 4) because it is difficult to know which value makes an individual acceptable. The user must establish for how long the algorithm be run. Finally, the algorithm returns the individual from the new population with the best fitness value (line 22).

4.2.2. Fitness function

The fitness function provides a method for quantitatively ranking individuals. By comparing fitness values, we can determinate whether one individual is better than another. The proposed fitness function contains two addends. The first addend is the land suitability that each plot has for the category allocated to it, and the second is the compactness of the resultant land use patches. We refer to the former as aptitude and the latter as compactness. Both addends are weighted to allow the user to establish the importance of one against the other. Equation 1 defines that.

$$Fitness = w_a * Aptitude + w_c * Compactness \tag{1}$$

where w_a and w_c are addend weights, and they are input parameters. They are normalized, and their sum must be 1.

Aptitude is calculated using the weighted average of the aptitudes for each category. Aptitude for a category is obtained from the average of the aptitudes that the plots allocated to that category have for it, weighted by the area of each plot and normalized by the total area assigned to the category:

$$Aptitude = \sum_{i=1}^{C} w_i \left(\frac{\sum_{j=1}^{N_i} ap_{ij} * a_{ij}}{\sum_{j=1}^{N_i} a_{ij}} \right) \tag{2}$$

where C is the number of categories, w_i is the weight of the *i-th* category (it is an input parameter), N_i is the number of plots allocated to the *i-th* category, ap_{ij} is the aptitude that the *j-th* plot allocated to the *i-th* category has for it, and a_{ij} is the area of the *j-th* plot allocated to the *i-th* category.

Compactness can be defined in different ways. Our proposal considers two definitions related to the above explanation: one based on patches, which are groups of adjacent plots with the same category, and the other based on categories, where plots are grouped into categories. Users can choose the function to use.

For compactness based on patches, the following expression defines the function:

$$Compactness_{patches} = 4\Pi \sum_{i=1}^{C} w_i \left(\frac{\sum_{j=1}^{NPa_i} \frac{a_{ij}}{p_{ij}^2}}{NPa_i} \right) \tag{3}$$

where NPa_i is the number of patches of the *i-th* category, p_{ij} and a_{ij} are the perimeter and area values of the *j-th* spot of the *i-th* category, respectively, and C and w_i have the same meaning as in (2).

This formula arises because, for a given area, a circle maximizes the so-called circularity (Montero and Bribiesca, 2009), where circularity is defined thus:

$$Circularity = 4\Pi \frac{area}{perimeter^2} \tag{4}$$

Function (3) has high computational costs, as the patches of each individual of the new population must be calculated in each iteration.

The compactness function based on categories is also based on (4) but avoids computing the patches:

$$Compactness_{categories} = 4\Pi \sum_{i=1}^{C} w_i \left(\frac{\sum_{j=1}^{N_i} a_{ij}}{(\sum_{j=1}^{N_i} p_{ij})^2} \right) \quad (5)$$

where C, w_i, N_i, and a_{ij} have the same meaning as in (2), and p_{ij} is the perimeter of the *j-th* plot allocated to the *i-th* category. This function presents clearly lower computational costs and, as shown below, produces good results.

The constraints in the size of each category implies: $min_i \leq \sum_{j=0}^{N_i} a_{ij} \leq max_i$, where min_i and max_i are the minimum and maximum area allowed for the $i-th$ category respectively.

5. Parallel genetic algorithm

The genetic algorithm execution times are high due to the large number of plots and the implicit nature of the problem. To get a practical algorithm, these execution times must be reduced: the solution lies in parallelizing the algorithm. Three parallel solutions were implemented: a multi-core solution based on the shared-memory paradigm, a cluster one based on the message passing paradigm and a hybrid one. These target systems are important, as they are currently widespread.

5.1. The parallel algorithm for multi-cores

Current computers, from servers to laptops and smart-phones, often have multi-core processors. It is interesting to implement algorithms to take advantage of this architecture.

We focus on parallelizing the genetic loop, i.e., the processes to generate new individuals through multiple threads. Each thread generates a set of individuals by performing selection, crossover, mutations and the election operation. As threads share memory, they can all read the same population and update the next population.

The number of threads is an input parameter. Given this number and the population size, the number of individuals to be created by each thread can be established. This distribution must be as balanced as possible in terms of execution time. Figure 3 shows an example of how the work distribution is performed. All threads can select any individual from the population,

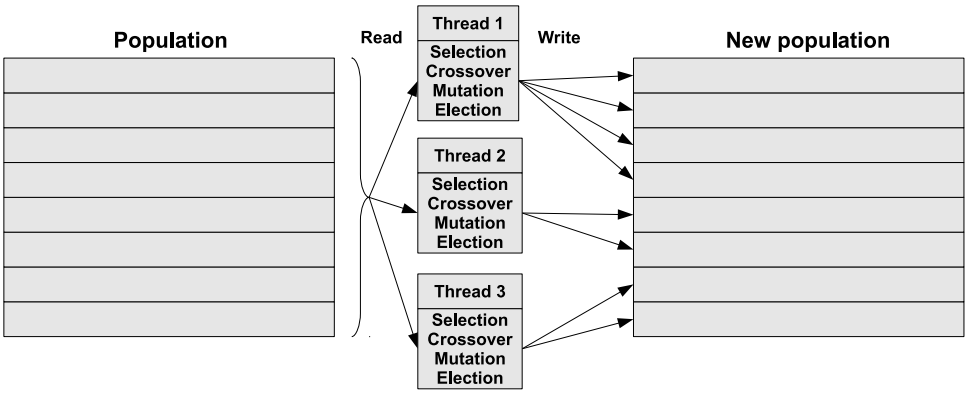

Figure 3: Workload among slave threads in the multi-core algorithm.

but they can only write individuals in the new population in the positions assigned to them.

Figure 4 shows the structure and stages of the multi-core parallel genetic algorithm.

The Java package java.util.concurrent (Java.com, 2011) is used to launch threads in different cores. For each genetic loop iteration, a master thread is in charge of starting all slave threads, and it waits until they finish. Each slave thread is independent of the rest, so the parallelization is straightforward and efficient. Once the master thread recovers control, the new population has been already created and master thread only checks the stop criterion.

With this parallel method there is no guarantee of obtaining individuals better than those in the same iteration of its sequential counterpart. Nevertheless, it is possible to run more iterations of the genetic loop and generate more individuals than the sequential algorithm in the same time. With this method, convergence can be reached earlier.

5.2. The parallel algorithm for clusters

Clusters are distributed memory systems, so processes must be able to send and receive messages to exchange information. An open-source Java message passing library called MPJ Express was thus used.

Each process is executed in a cluster node, and a master process is in charge of establishing communications with the slave processes. Each slave executes the whole genetic algorithm independently from the other slaves, using its own population. Synchronization with the master process is performed periodically. This parameter is set by the user, who can choose a

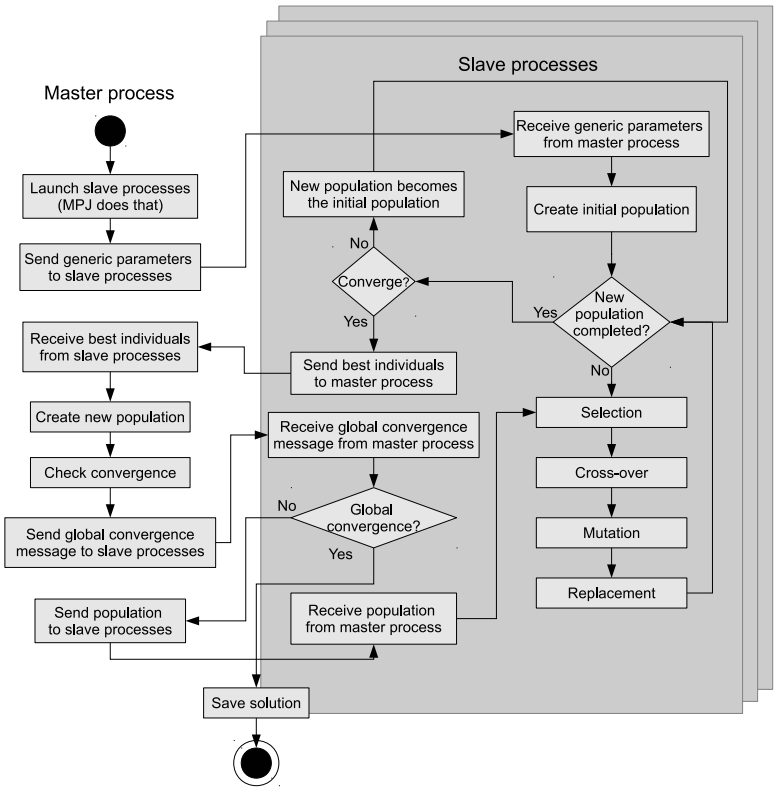

Figure 4: Multi-core parallel genetic algorithm work flow.

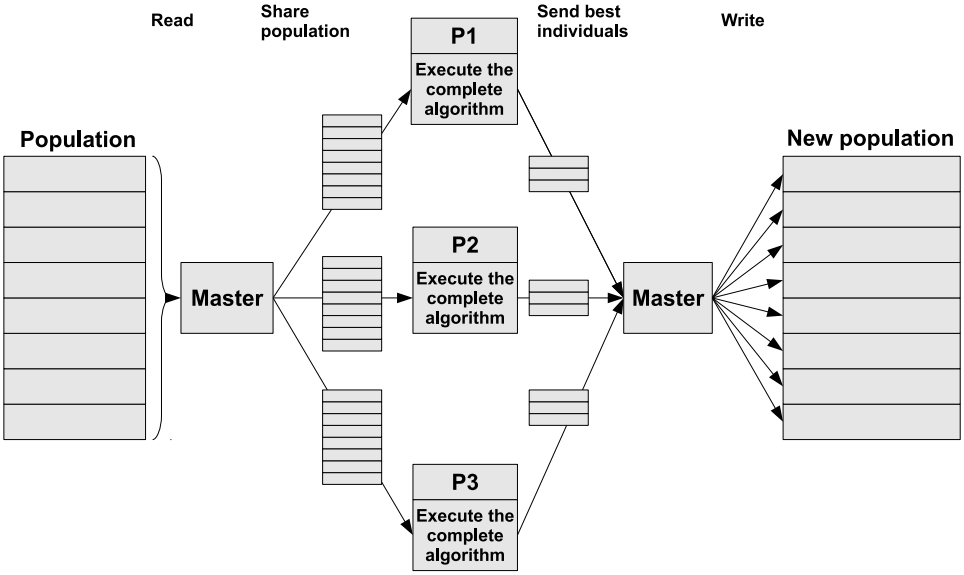

Figure 5: Workload among slave processes in the cluster algorithm.

period of seconds, minutes or even hours. This synchronization consists of sending the best individuals from the slaves to the master. After gathering this information, the master creates a new population. This new population is then broadcast to the slaves to start the algorithm again. The master checks the stop criterion. Figure 5 shows a summary of this parallel implementation.

This version takes advantage of each slave process generating its own population at each genetic loop iteration, and it executes the whole algorithm until synchronization. After synchronization, the slaves receive the same population that will evolve in their own way because of the randomness of the genetic operators. The best individuals obtained from executing different slaves also join the new population, raising the average fitness and increasing the search space.

An interesting issue is that implementing the algorithm allows slave processes to read their own local parameters files. In this way, different processes can work with different parameters, e.g. crossover and mutation rates, and selecting new populations after synchronizations provides a richer set of individuals, thus achieving more heterogeneity and potentially better results.

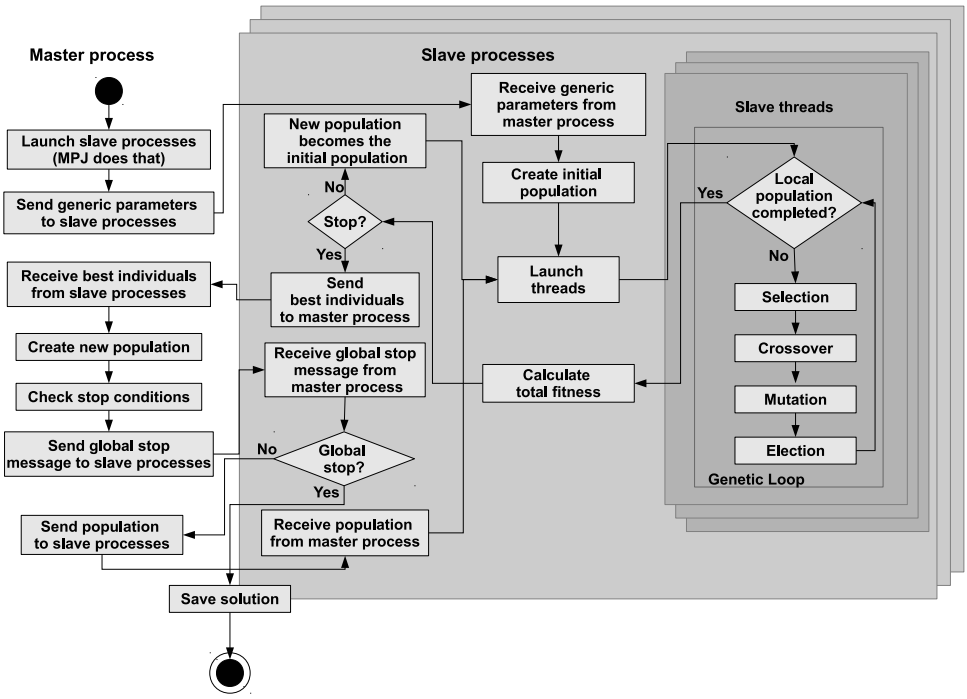

Figure 6: Hybrid parallel genetic algorithm work flow.

5.3. The hybrid algorithm for multi-core clusters

The previous sections proposed two different parallelism environments. These environments are orthogonal in that they can be mixed to produce a new algorithm that can be executed in a cluster with multi-core nodes taking advantage of both parallel paradigms. As shown below, this implementation presents better results in the fitness of the final result, as it exploits multiple parallelism levels.

Figure 6 shows the communications between the master and the slave processes and how they launch local slave threads.

6. Experimental results

For our study, some tests were performed in a Galician municipality called Guitiriz, with 138,175 plots, 52,045 of which have a fixed category. The individuals thus have 86,130 genes: 10,232 are fixed plots, so the genetic algorithm considers the 75,898 remaining plots as candidates for category

changes. For this municipality, four non-fixed categories are considered: natural area, agricultural, forestry, and urban. All performance tests were executed in a cluster node with two Intel Xeon E5520 processors and 8GB RAM. Each processor has 4 cores at 2.27GHz provided with hyper-threading (two threads per core are allowed to run simultaneously). Using this cluster was shared with other users, so some variability should be considered regarding the measured execution times.

6.1. Influence of population size

As a representative case, the results of Figure 7 correspond to a set of multi-core parallel executions over 16 threads and 32, 64, 128 and 256 individuals in the population. This figure shows the average of the values obtained from running the algorithm 20 times with each population size for three hours. The fitness function weights were 0.25 for the aptitude and 0.75 for the compactness. With small populations, the algorithm runs more iterations per unit time and achieves better results in the short-term than the executions with greater population sizes. However, in the long-term (24 hours for the 256 individuals executions, not shown in Figure 7), our tests indicate that the fitness tends to be slightly better with high values for the population size, presumably because of the increased explored search space.

6.2. Influence of the mutation rate

Some tests to find the best mutation rate (MR) were also executed. Figure 8 shows the average of five cases executed 20 times each with a population of 32 individuals and different mutation rates, using the recommended values mentioned in Section 4.2.1. Each execution lasted one hour. These tests also include an adaptive mutation rate (AMR), which starts with a high value and reduces the MR (75% in the tests) to change fewer genes and adjust precision when the best fitness of the population does not improve in a certain number of iterations (20 in the tests).

These results indicate that executions with high mutation rates perform poorly. This occurs because too many changes in individuals' genes make the mutation behaviour almost random. If the mutation rate proposed by Grefenstette is used (1/L, with L the individual size, that in our case is $1.3174E^{-5}$), then the algorithm obtains the best individuals. Comparing results obtained using MR and AMR individuals tends to be similar in the long-term, but at least theoretically, a larger search space is explored using an AMR.

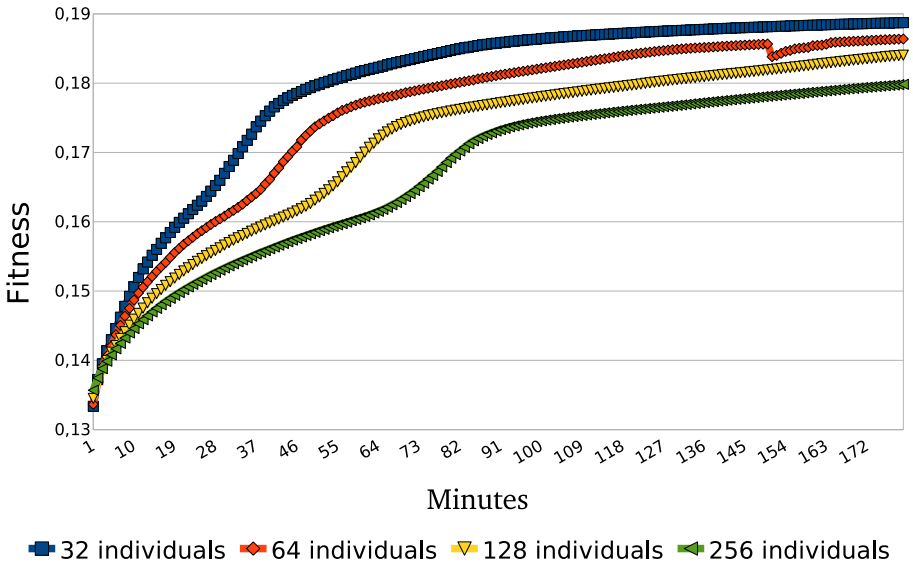

Figure 7: Fitness versus the population size.

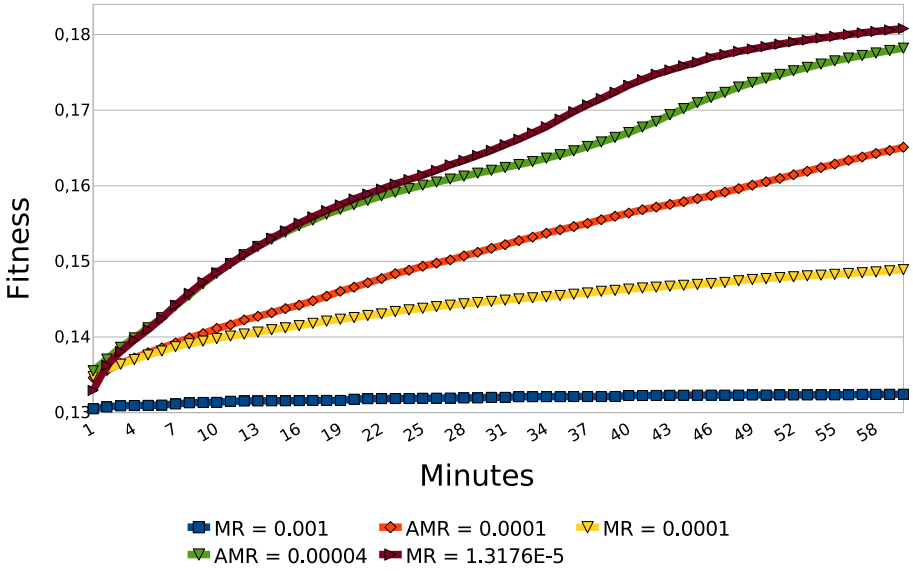

Figure 8: Aptitude versus static mutation rate and adaptive mutation rate.

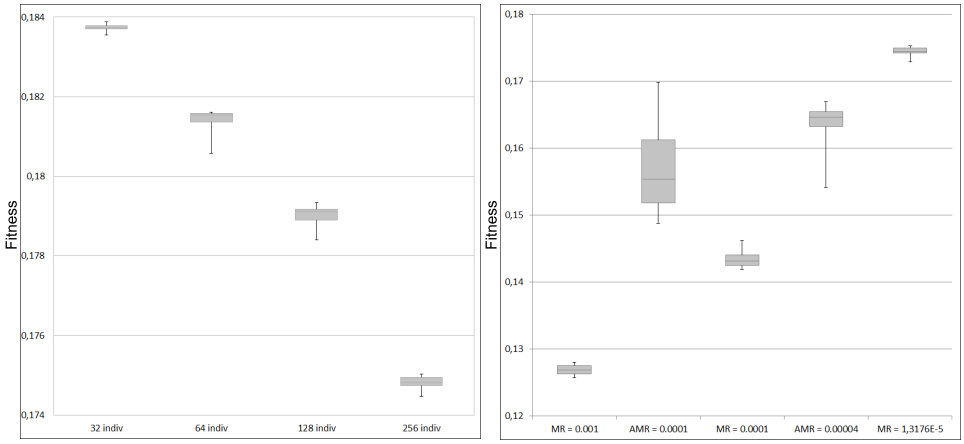

Figure 9: Box plots of the population size (A) and mutation rate (B) tests.

Figure 9 depicts the variance of the set of results obtained in the populations size and mutation rate tests. Note that the values are quite homogeneous, which prove that the use of the average as representative data in the plots, is a valid method of measure.

6.3. Comparison between fitness functions

As mentioned above, two compactness functions were implemented: one based on patches, and another one based on categories. Aiming to study the effects of both functions, a rectangular parcel map was created. This parcel map has 10,496 plots with different sizes, and each plot has only one suitable category (the aptitude is one for this category and zero for the rest). The same list of categories as the real case (see the beginning of section 6) is used.

These tests were executed with the multi-core version of the algorithm using 16 threads in a 8 cores machine with hyperthreading for three hours. Tables 3 and 4 show the average of five executions for each test. The columns show the values of the different fitness function components for the corresponding configuration: *Apt* shows the aptitude value, *CC* the compactness value by category, and *CP* the compactness value by patches. The number of patches obtained (NPa) and percentage of the plots allocated to the right category (hits) are also shown.

The rows indicate the configuration with which the algorithm was executed: label *Aptitude* means that the fitness function only considers aptitude; labels *Comp. Cats.* and *Comp. Pats.* mean that the fitness func-

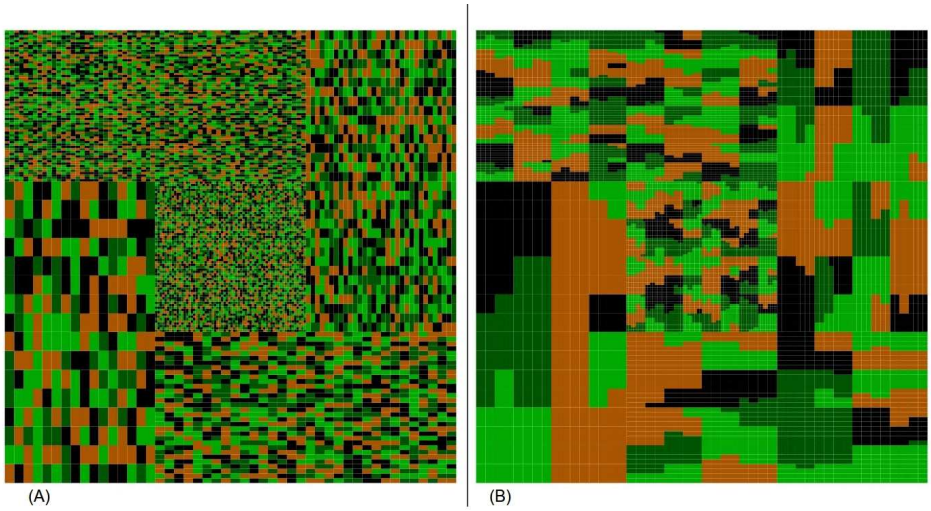

Figure 10: Plot maps: random allocation (A) and compacted one (B).

tion only considers the compactness by categories and patches, respectively; *Apt:CC X%:Y%* and *Apt:CP X%:Y%* mean that the fitness function considers both attributes, i.e., aptitude at X% and compactness (using categories and patches, respectively) at Y%; and *CC+CP X%:Y%* mean that both compactness functions are considered, i.e., using categories at X% and patches at Y%. This last configuration was added after testing the other configurations. As the *CC* configurations reduce the number of patches and the *CP* configurations help make them more compacted, a new fitness function was implemented integrating both compactness methods as two addends.

The difference between the tables is that the aptitudes for the plot map were randomly set in Table 3 (hereafter the *random allocation*), and the aptitudes were set with some compactness (*compacted allocation*) in Table 4, as in Figure 10. This figure shows the aptitude-based optimal plot map, as the best category is allocated to each plot. Compacted allocation is closer to reality.

Tables 3 and 4 show that when only the aptitude is used, our algorithm achieves 100% of hits (first row in both tables). In real cases, the aptitude value of the categories is neither one nor zero but takes intermediate values, so it is more difficult to reach 100% of hits.

With the aptitude random allocation (Table 3) and using only compactness by categories (second row), the number of patches between executions

	Apt	CC	CP	NPa	Hits
Aptitude (Apt)	1	0.000633	0.6655	5387	100%
Comp. Cats. (CC)	0.3005	0.0555	0.6392	143	28.14%
Apt:CC 25%:75%	0.3229	0.0408	0.6022	110	29.53%
Apt:CC 50%:50%	1	0.000633	0.6655	5386	99.99%
Apt:CC 75%:25%	1	0.000633	0.6655	5387	100%
Comp. Pats. (CP)	0.4972	0.000816	0.7787	6244	48.70%
Apt:CP 25%:75%	0.9095	0.000604	0.7509	7065	78.45%
Apt:CP 50%:50%	0.9712	0.000572	0.7172	6934	91.04%
Apt:CP 75%:25%	0.9992	0.000631	0.6677	5445	99.76%
CC+CP 50%:50%	0.3374	0.0191	0.6975	327	33.20%

Figure 3: Comparison between fitness functions with random aptitudes.

can change considerably (in our tests, from 53 to 213 patches, 143 is the average). This occurs because each execution starts with a random solution, and the groups of categories obtained thus differ. As the aptitude is ignored in this case, the evolution of the individuals can be quite different. If 25% of the aptitude is added to the fitness function, that variance is reduced (from 73 patches as minimum to 211 as maximum) and the number of hits slightly increases. With a fifty-fifty balance between aptitude and compactness, the algorithm almost reaches 100% of hits. With the same compactness method in the fitness function but using the compacted allocation as a plot map (Table 4), less compactness is obtained as more aptitude is added, but more hits are achieved and the number of patches is closer to the aptitude-based optimal plot map.

When using only compactness by patches, many patches are created. This

	Apt	CC	CP	NPa	Hits
Aptitude (Apt)	1	0.0138	0.4620	165	100%
Comp. Cats. (CC)	0.6172	0.0494	0.5739	65	52.07%
Apt:CC 25%:75%	0.6187	0.0505	0.5863	68	52.85%
Apt:CC 50%:50%	0.7051	0.043	0.0.5691	73	61.41%
Apt:CC 75%:25%	0,9387	0.0169	0.502	137	92.73%
Comp. Pats. (CP)	0.6908	0.0021	0.7795	4030	56.39%
Apt:CP 25%:75%	0.95	0.0042	0.7728	2747	83.44%
Apt:CP 50%:50%	0.9716	0.006	0.7639	1769	89.60%
Apt:CP 75%:25%	0.9841	0.0078	0.7481	1153	93.79%
CC+CP 50%:50%	0.7636	0.0264	0.6330	144	72.22%

Figure 4: Comparison between fitness functions with compact aptitudes.

happens using any allocation method in the plot map (as in the bottom half of Tables 3 and 4). If both compactness functions are combined, a good relation among CC, CP, and NPa is obtained because compactness by categories tends to create fewer patches, and compactness by patches helps make them more compacted. If the goal is more hits, i.e., if it is more important to choose the suitable category than achieve a good compactness degree, a percentage of aptitude can be added to the fitness function. In the results with random allocation, the number of patches obtained was varied from 104 to 419. With compactness allocation, however, this variance was reduced (from 136 to 160 patches).

Figure 11 shows the results obtained executing the *CC+CP:50%-50%* configuration using both plot maps of Figure 10. As in Tables 3 and 4, this is the best configuration to reach an agreement between all measured pa-

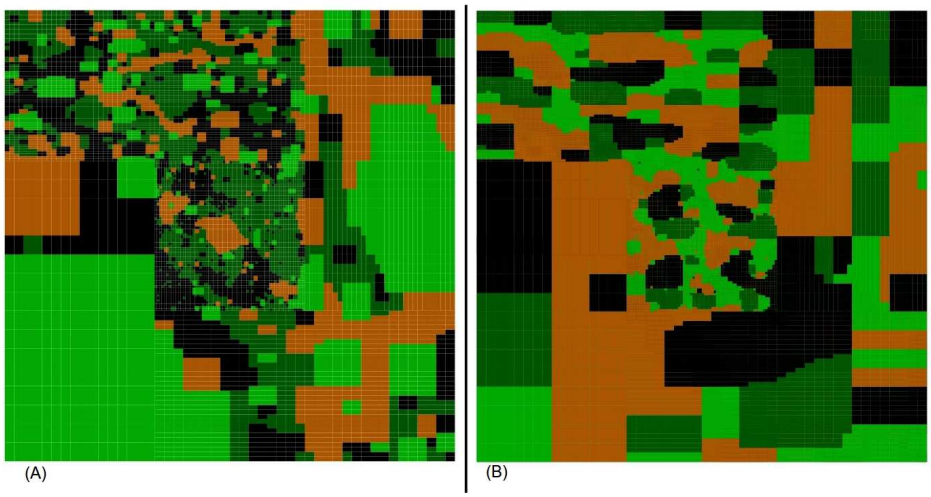

Figure 11: Plot map results for the *CC+CP:50%-50%* configuration: random allocation (A) and compacted one (B).

rameters. Comparing both solutions, the one using the compacted allocation (Table 4) as a plot map has fewer difficulties in achieving a compacted solution with few patches but many hits. As real cases often have that aptitude distribution, the results can be considered satisfactory.

Finally, as shown in a real case in section 6.5, these tables can help decide fitness function weights depending on the goals the planning expert wants to reach: aptitude, compactness, number of patches, or number of hits.

6.4. Performance results

Figure 12 shows the speed-up evolution for the multi-core parallel versions versus the sequential version (a single-threaded execution disabling core hyperthreading), and Figure 13 shows speed-up (A) and efficiency (B) for a particular case, considering 20,000 iterations. The parallel algorithm executions present super-linear speed-up, while the number of threads used by the parallel genetic algorithm is less than or equal to the number of physical computer cores, 4 in this case. This occurs due to a better memory hierarchy utilization, as each thread must deal with less data. Though speed-up still increases, efficiency decreases using more cores, as hyper-threading is not as efficient as using a different physical core for each thread (as expected according to the reports of the proper microprocessor manufacturers). Hyper-threading allows executing more than one thread simultaneously; as

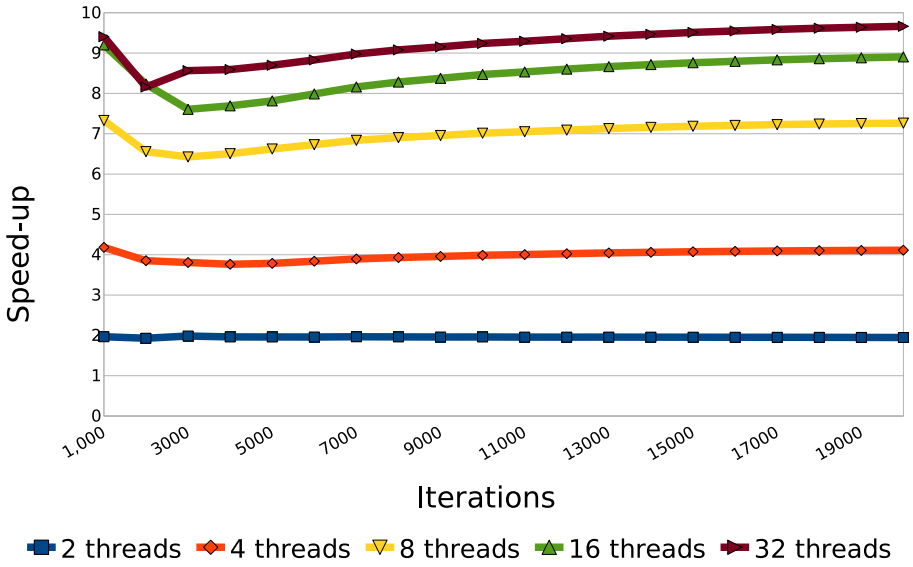

Figure 12: Speed-up for the multi-core parallel versions against the sequential one.

threads share some cpre resources, real performance only increases by 30-40 per cent at most. The charts shown in these two figures were obtained from the average of 20 executions performed for each test.

The speed-up for the cluster algorithm version is not shown, because that parallelization method aims to improve fitness in the shortest time. In this case, the speed-up of the slower node conditions the speed-up.

Figure 14 shows the fitness of the hybrid version versus the multi-core version. For these tests, 16 threads were used per node for both algorithms versions. The hybrid version was executed in two, four and eight nodes with different synchronization times. All executions using the hybrid version provided better results than those using the multi-core version.

Focusing on the hybrid algorithm executions, Figure 14 shows how the best results are obtained using 2 nodes with 15 minutes between synchronizations and 4 nodes with 6 minutes. The curve of the executions with 2 nodes tends to stand still, but the curve of the executions with 4 nodes (specially with 6-minutes synchronizations) has a steeper slope than the 2-node executions. Nevertheless, the differences between fitness among these executions are small. Executions with 8 nodes achieve lower fitness than the other

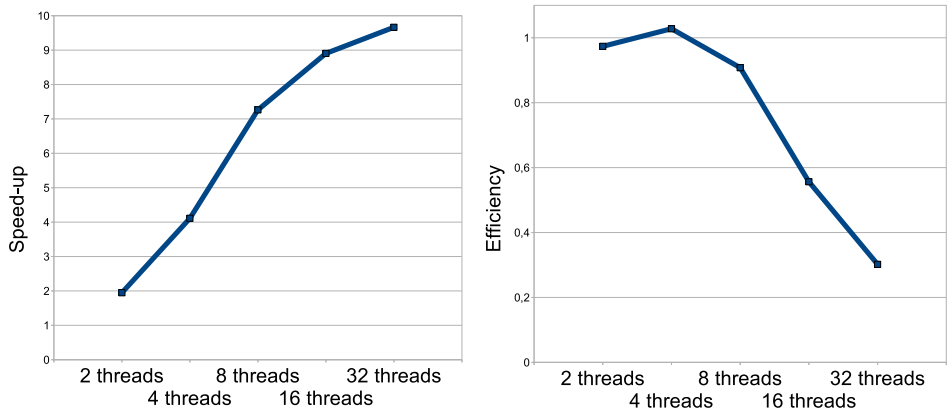

Figure 13: Speed-up (A) and efficiency (B) in the iteration 20,000 of the multi-core parallel versions against the sequential one.

executions, but longer executions may achieve better results. As mentioned above, the stop criterion was based on execution time, so these tests were executed during 3 hours to see the results clearly.

6.5. Real case test

Figure 15 shows a solution provided by the algorithm for the Guitiriz test case. This municipality has 138,175 plots, 52,045 of which have a fixed category. Knowing that the plot map of Guitiriz has some compactness in their aptitudes, according to the Table 4, tests were executed using the Apt:CC 50%:50% to get a good relation between the number of patches and hits. As the different colors indicate, plot patches form the expected regular polygons. White patches correspond to fixed categories, like riversides or roads. Other patches include the remaining plots, fixed and not-fixed.

Figure 16 compares the solution generated by the algorithm and one provided by experts. The first solution has more white patches (fixed categories), because expertos do not always account for protection zones around rivers or roads. Apart from this, both solutions have considerable similarities, and the algorithm solution is a good starting point for the experts.

7. Additional functionalities

Some functionalities are included in the algorithm implementation to help users and handle possible system failures.

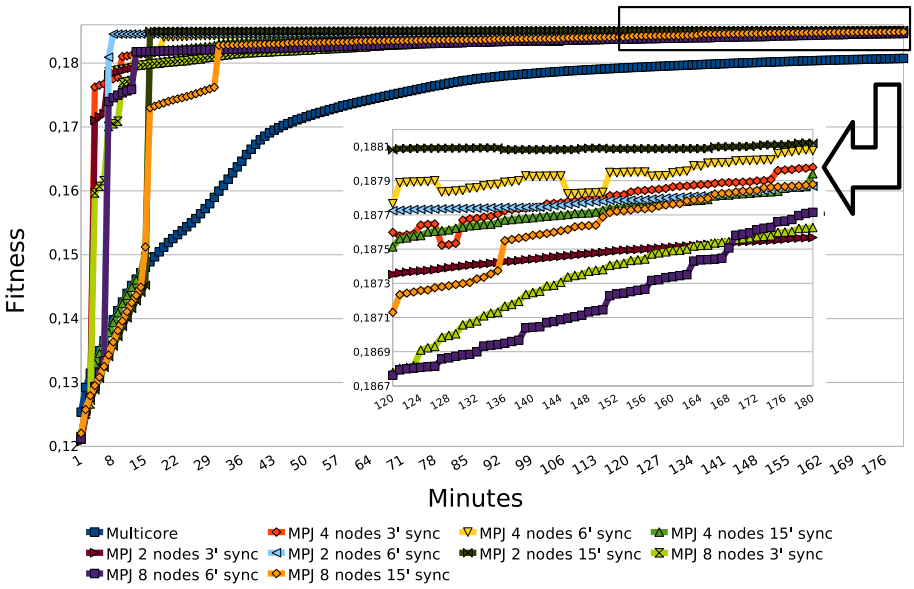

Figure 14: Fitness value for the multi-core version against the hybrid version.

7.1. Checkpointing

A mechanism to continue execution after possible system breakdowns was implemented, based on the idea of periodically saving the algorithm state. All individuals of the current population, including their fitness and some additional general information, are saved. If the algorithm stops for any reason, it can thus be rerun starting from the last stored data. Depending on the execution mode, i.e., sequential or parallel versions, the checkpointing was implemented in different ways. When the algorithm finishes each genetic loop iteration, it checks whether it is time to save the checkpoint file. In the cluster/hybrid parallel model, the master process is in charge of performing the checkpoint right after each synchronization. As the amount of information to be stored in the checkpointing files can be large, these files are compressed.

7.2. Saving the best individual

After every iteration in the sequential and multi-core algorithm versions, the best individual from the entire population is saved. At the end of the algorithm, the solution is the best individual obtained at any iteration during the execution. Regarding the cluster and hybrid versions, this operation

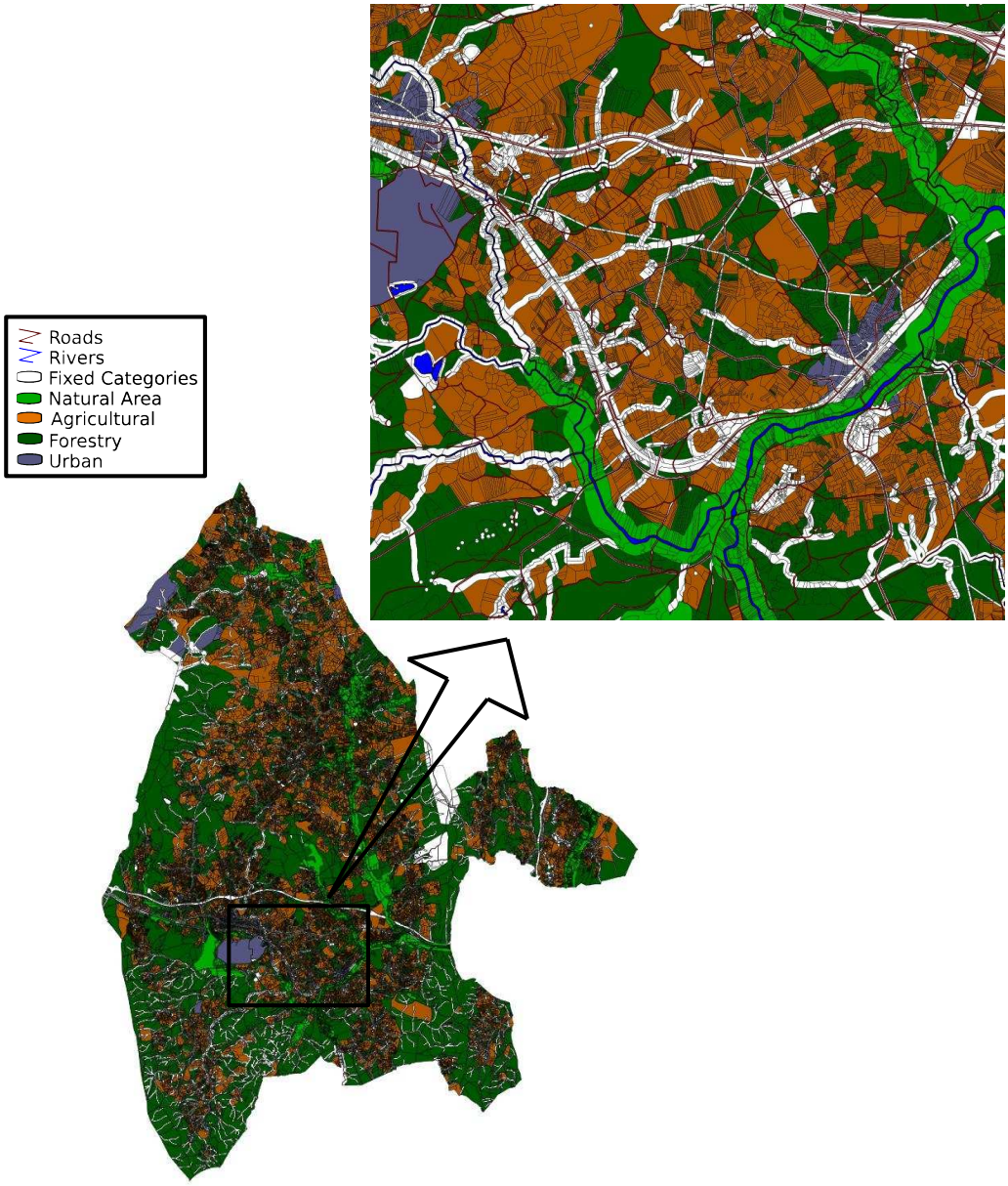

Figure 15: Algorithm output.

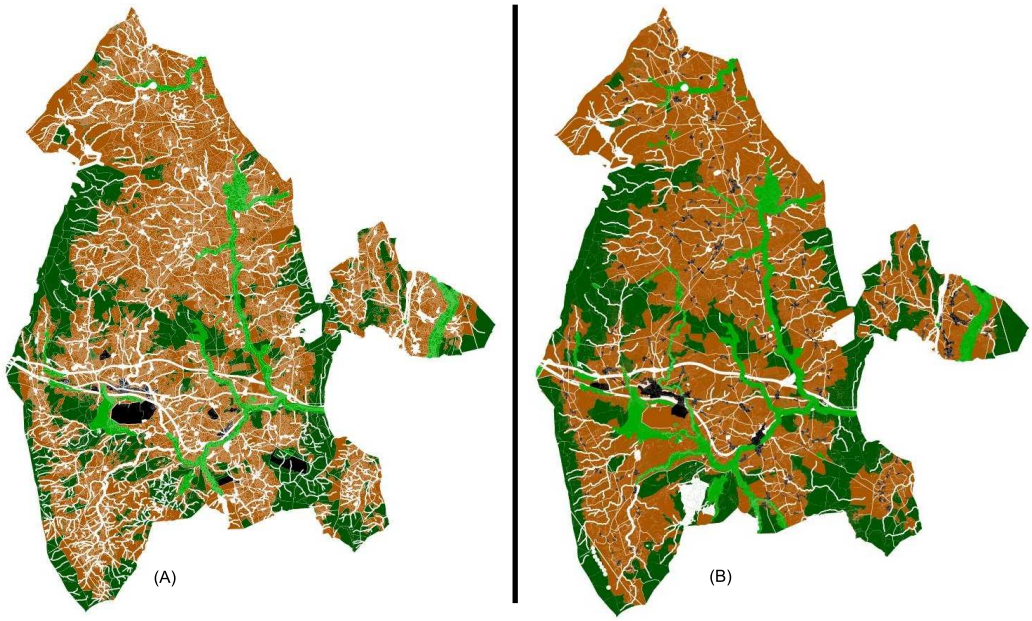

Figure 16: Comparison between algorithm's solution (A) and experts' solution (B).

is performed after every synchronization. Every slave also saves the best individual, and they are sent to the master process at the synchronization point.

7.3. User interfaces

To implement the algorithm, a Java-free geospatial analysis tool called SEXTANTE (Olaya, 2011) was used. Its main aim is being a platform to develop geoalgorithms that eases both the implementation and use of these algorithms. SEXTANTE can be used in desktop GIS software like uDig, openJUMP or gvSIG through their GUI (Graphical User Interface) or command-line interface. The planner can use the GUI of our algorithm in sequential or multi-core computers and the command-line interface in clusters where graphical user interfaces cannot be displayed. Figure 17 shows the GUI for gvSIG.

8. Conclusions and future work

This work shows that parallel genetic algorithms are a good choice to deal with land use planning problems where the number of possible plot and

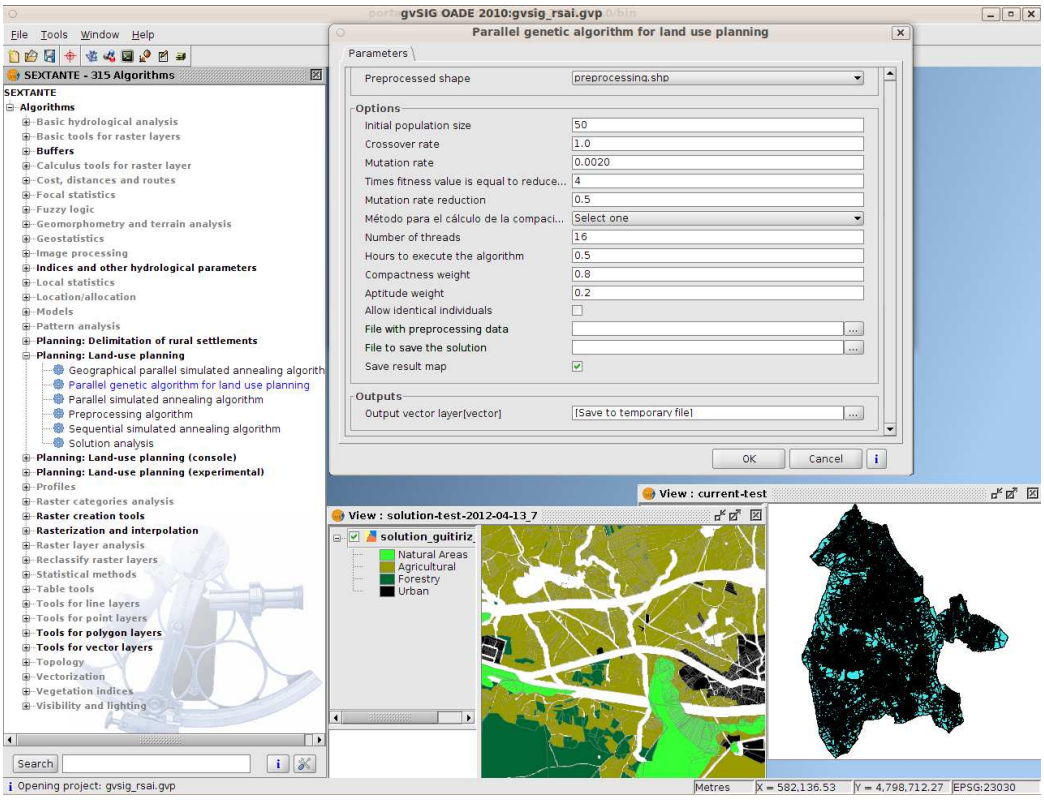

Figure 17: GUI for gvSIG with SEXTANTE.

category combinations could be huge. This work has reached several conclusions. One conclusion is that this kind of algorithm can be parallelized and executed in multi-core systems where it can reach an almost linear speedup. In cluster systems, the algorithm takes advantage of multiple parallel executions. If the cluster contains multi-core nodes, both parallelization strategies can be merged, resulting in a parallel hybrid version of the algorithm.

Several tests were executed to analyze the algorithm behaviour, depending on the plot map or attributes considered for the fitness function. The results of these tests were discussed and shown to be a good starting point for the planning experts. Experts can use the results in Tables 3 and 4 to configure the algorithm to obtain better results according to their goals.

Finally, developing efficient parallel Java code has proven to be a competitive solution: programmability issues are clearly in favor of Java, where acceptable performance has been obtained.

As future work, one of the most immediate improvements is to study other functions to evaluate compactness criteria. In particular, patches of the same category are usually separated by roads or water bodies, which should be considered to avoid diminishing the compactness value. Other optimization criteria can also be considered, including favoring the creation for patches of agricultural uses next to the localization of existing farms. Creating corridors to connect patches of natural protected areas should also be promoted. Another idea for future work is applying new parallelism levels, e.g., in evaluating the fitness function. Finally, we are working on developing algorithms to limit growth areas for rural settlements. Comparing the results of the genetic algorithm with an implementation based on simulated annealing is also in progress.

Acknowledgements

This work is included in the project named *Geographical Information Systems for Urban Planning and Land Management using Optimization Techniques on Multi-core Processors* (ref. 08SIN011291PR), supported by the projects of *Consolidation of Competitive Research Groups* (ref. 2010/06 and 2010/28), all funded by the Galician Regional Government, Spain.

References

Aerts, J. C. J. H., Eisinger, E., Heuvelink, G. B. M., Stewart, T., 2003. Using linear integer programming for multi-site land-use allocation. Geographical

Analysis 35 (2), 148–169.

Aerts, J. C. J. H., van Herwijnen, M., Janssen, R., Stewart, T. J., 2005. Evaluating spatial design techniques for solving land-use allocation problems. Journal of Environmental Planning and Management 48 (1), 121–142.

Arentze, T. A., Borgers, A. W. J., Ma, L., Timmermans, H. J. P., 2010. An agent-based heuristic method for generating land-use plans in urban planning. Environment and Planning B: Planning and Design 37, 463–482.

Armstrong, M. P., Densham, P. J., 1992. Domain decomposition for parallel processing of spatial problems. Computers, Environment and Urban Systems 16, 497–513.

Balling, R. J., Taber, J. T., Brown, M. R., Day, K., 1999. Multiobjective urban planning using genetic algorithm. Journal of Urban Planning and Development 125 (2), 16–99.

Brookes, C. J., 2001. A genetic algorithm for designing optimal patch configurations in gis. International Journal of Geographical Information Science 15 (6), 539–559.

Bäck, T., 1996. Evolutionary Algorithms in Theory and Practice. Oxford Univ. Press.

Cromley, R. G., Hanink, D. M., 1999. Coupling land use allocation models with raster gis. Journal of Geographical Systems 1, 137–153.

Duh, J. D., Brown, D. G., 2007. Knowledge-informed pareto simulated annealing for multi-objective spatial allocation. Computers, Environment and Urban Systems 31, 235–281.

Eastman, J., Jin, W., Kyem, P. A. K., Toledano, J., 1995. Raster procedures for multi-criteria/multi-objective decisions. Photogrammetric Engineering & Remote Sensing 61 (5), 539–547.

Ehrgott, M., Gandibleux, X., 2000. A survey and annoted bibliography of multiobjective combinatorial optimization. OR Spektrum.

Eldrandaly, K., 2010. A gep-based spatial decision support system for multisite land use allocation. Applied Soft Computing 10, 694–702.

FAO et al., 1993. Guidelines for Land-Use Planning. Development Documents Series. Food and Agriculture Organization of the United Nations.

Feng, C. M., Lin, J. J., 1999. Using a genetic algorithm to generate alternative sketch maps for urban planning. Computers, Environment and Urban Systems 23, 91–108.

Goldberg, D. E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning, 1st Edition. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Goldberg, D. E., Deb, K., 1991. A comparative analysis of selection schemes used in genetic algorithms. In: Foundations of Genetic Algorithms. Morgan Kaufmann, pp. 69–93.

Grefenstette, J. J., jan 1986. Optimization of control parameters for genetic algorithms. Systems, Man and Cybernetics, IEEE Transactions on 16 (1), 122 –128.

Guan, Q., Clarke, K. C., 2010. A general purpose parallel raster processing programming library test application using a geographic cellular automata model. International Journal of Geographical Information Science 24 (5), 695–722.

Holzkamper, A., Seppelt, R., 2007. A generic tool for optimizing land-use patterns and landscape structures. Environmental Modelling & Software 22, 1801–1804.

Huang, H., Wei, Z., Li, Z., 2009. The Geographic Information System Based on Distributed Parallel Computation. Networking and Digital Society, ICNDS '09 International Conference 1, 234–237, 30-31 May 2009.

Janssen, R., van Herwijnen, M., Stewart, T. J., Aerts, J. C. J. H., 2008. Multiobjective decision support for land-use planning. Environment and Planning B: Planning and Design 35, 740–756.

Java.com, 2011. http://download.oracle.com/javase/6/docs/api/java/util/concurrent/package-summary.html, visited on 31 July 2011.

Jong, K. D., 1975. An analysis of the behavior of a class of genetic adaptive systems. Ph.D. thesis, University of Michigan, Ann Arbor, MI, USA.

Kai, C., Bo, H., Qing, Z., Shengxiao, W., aug 2009. Land use allocation optimization towards sustainable development based on genetic algorithm. In: Geoinformatics, 2009 17th International Conference on. pp. 1 –5.

Lei 2/2002, 2002. Galician Official Diary 252, 18,025 – 18,094, Lei 9/2002, do 30 de decembro, de ordenación urbanística e protección do medio rural de Galicia (in Galician).

Lei 2/2010, 2010. Galician Official Diary 61, 4,639 – 4,666, Lei 2/2010, do 25 de marzo, de medidas urxentes de modificación da Lei 9/2002, do 30 de decembro, de ordenación urbanística e protección do medio rural de Galicia (in Galician).

Matthews, K. B., Sibbald, A. R., Craw, S., 1999. Implementation of a spatial decision support system for rural land use planning: integrating geographic information system and environmental models with search and optimisation algorithms. Computers and Electronics in Agriculture 23 (1), 9 – 26.

Montero, R. S., Bribiesca, E., 2009. State of the art of compactness and circularity measures. International Mathematical Forum 4 (25-28), 1305–1335.

Nyerges, T., Jankowski, P., 2009. Regional and urban GIS: a decision support approach. Guilford Press.

Olaya, V., 2011. www.sextantegis.com, visited on 31 July 2011.

Porta, J., Parapar, J., Doallo, R., Barbosa, V., Santé, I., Crecente, R., 2012. Evolutionary Algorithm for the Demarcation of Rural Settlements. In: 9th World Congress of the Regional Science Association International. Timisoara, Romania. Paper accepted.

Santé-Riveira, I., Crecente-Maseda, R., Miranda-Barrós, D., 2008. GIS-based planning support system for rural land-use allocation. Computers and Electronics in Agriculture 63 (2), 257 – 273.

Shafi, A., 2011. http://mpj-express.org/, visited on 31 July 2011.

Spears, W. M., DeJong, K., 1991. An analysis of multipoint crossover. In: Proc. of Workshop of the Foundations of Genetic Algorithms. pp. 301 – 315.

Stewart, T. J., Janssen, R., Herwijnen, M., 2004. A genetic algorithm approach to multiobjective land use planning. Computers & Operations Research 31, 2293–2313.

Taboada, G. L., Touriño, J., Doallo, R., 2009. Java for High Performance Computing: Assessment of Current Research and Practice. In: 7th International Conference on the Principles and Practice of Programming in Java, PPPJ 2009. ACM International Conference Proceeding Series. Calgary, Alberta, Canada, pp. 30–39.

Xiao, N., Bennett, D. A., Armstrong, M. P., 2001. Interactive evolutionary approaches to multiobjective spatial decision making: A synthetic review. Computers, Environment and Urban Systems 31, 232–252.

Xibao, X., Jianming, Z., Xiaojian, Z., 1995. Integrating gis, cellular automata and genetic algorithm in urban spatial optimization - a case study of lanzhou. In: Proc of SPIE. Vol. 6420. 64201U-1 - 64201U-10.

Xin, H., Zhi-xia, Z., 2008. Application of genetic algorithm to spatial distribution in urban planning. In: IEEE International Symposium on Knowledge Acquisition and Modeling Workshop. pp. 1026–1029, wuhan, China.

Zhang, H. H., Zeng, Y. N., Bian, L., 2010. Simulating multi-objective spatial optimization allocation of land use based on the integration of multi-agent system and genetic algorithm. International Journal of Environmental Research 4 (4), 765–776.