Stat Comput (2006) 16:251–260
DOI 10.1007/s11222-006-8079-7

# Comparative analysis of modern optimization tools for the *p*-median problem

**Enrique Alba · Enrique Domínguez**

Received: July 2004 / Accepted: February 2006


**Abstract** This paper develops a study on different modern optimization techniques to solve the *p*-median problem. We analyze the behavior of a class of evolutionary algorithm (EA) known as cellular EA (cEA), and compare it against a tailored neural network model and against a canonical genetic algorithm for optimization of the *p*-median problem. We also compare against existing approaches including variable neighborhood search and parallel scatter search, and show their relative performances on a large set of problem instances. Our conclusions state the advantages of using a cEA: wide applicability, low implementation effort and high accuracy. In addition, the neural network model shows up as being the more accurate tool at the price of a narrow applicability and larger customization effort.

**Keywords** Evolutionary algorithms · Cellular genetic algorithms · Neural networks · Optimization tools · *p*-median

## 1. Introduction

Solving NP-hard optimization problems is a core research area for many communities in engineering, operations research and computer science. The interdisciplinary features of most NP-hard problems have caused a large amount of contributions in the past. Researchers have proposed general tailored algorithms to overcome the many difficulties of medium and large sized instances of such problems. In this work, we have selected the *p*-median problem as our case of study.

The *p*-median problem is a discrete location-allocation problem belonging to a larger class of problems known as *p*-selection problems, where the solutions are generated by selecting *p* items from a finite universe. In the *p*-median case, the objective is to select, from a candidate set of facility points, *p* locations for such facilities in a way that the sum of some distance criterion from the set of users to the chosen facility points is minimised.

In this problem, most services are provided by desirable or non-obnoxious facilities, such as warehouse, shops, supermarkets, banks, or garages, where it is beneficial for the facilities to be located close to the customers they will be serving. Transportation costs and environmental impact (considered also as a cost) are assumed to be linear functions of the distance between the facility and the population centers or customers.

Kariv and Hakimi (1979) showed that the *p*-median problem on a general network is N*P*-hard. A number of solution procedures have been developed for general networks. Numerous techniques are based on mathematical programming relaxation and branch-and-bound algorithms. Also, many heuristics exist for this problem, such as tabu search (Rolland *et al.*, 1996), genetic algorithms (Nogueira and Furtado, 2001), neural networks (Domínguez and Muñoz, 2002) or scatter search (García-López *et al.*, 2003).

Complex problem solving usually means dealing with computationally hard tractable solutions (Garey and Johnson, 1979). In the past few years, several researchers used algorithms based on a model of organic evolution as an attempt to solve hard optimization and adaptation problems (Holland, 1975). Due to their representation scheme for search points, genetic algorithms (GA) (Goldberg, 1989) are one of the most promising and easily applicable

E. Alba (✉) · E. Domínguez
University of Málaga, Spain
e-mail: eat@lcc.uma.es

E. Domínguez
e-mail: enriqued@lcc.uma.es

representatives of evolutionary algorithms (EA) for the problem discussed in this paper. It has also been traditional to use neural networks (NN) to solve optimisation problems. This work addresses both EAs and NNs as competing paradigms for the analysis.

This work makes several contributions. First, a performance comparison between several evolutionary algorithms is made for solving the *p*-median problem. The considered algorithms are: a generational genetic algorithm (genGA) (Syswerda, 1991), a cellular genetic algorithm (cGA) (Manderick and Spiessens, 1989), and other evolutionary algorithms of reported efficiency and accuracy drawn from literature (i.e., problem dependent). Second, we can report that a single and simple cellular GA has the same or better accuracy than several state-of-the-art algorithms for large problem instances. The same algorithm (cellular) will be shown to outperform in either efficiency or accuracy, other sophisticated techniques including parallelism and local search. Last, but not least, we include a customized neural model that has still higher accuracy.

The outline of the paper is as follows. Section 2 presents the *p*-median problem. Section 3 presents an overview of the working principles of genetic algorithms. The encoding, the fitness function, and other specific problem-solving information are explained in this section. The experimental results for each problem instance are discussed in Section 4. We summarise our findings in Section 5.

## 2. Problem formulation

The well known *p*-median selection problem has been studied for many years. A generic selection problem consists of choosing the set of items that minimises a cost function subject to some constraints. In a *p*-selection problem, all the feasible solutions have size *p*. Several of the most relevant combinatorial optimization problems can be formulated as *p*-selection problems for an appropriate *p*, such as the travelling salesman problem (TSP), the Knapsack problem, finding the minimal Spanning Tree, the Steiner problem, and the *p*-median problem.

We focus on the *p*-median problem, which is concerned with the location of *p* facilities (*medians*) to minimise the total weighted distance between the facilities and the demand points. ReVelle and Swain (1970) provided an integer programming formulation for the discrete *p*-median problem, which is given below:

$$\text{Minimise} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} \tag{1}$$

$$\text{Subject to :} \quad \sum_{j=1}^{n} x_{ij} = 1 \quad i = 1,\ldots n \tag{2}$$

$$\sum_{j=1}^{n} x_{jj} = p \tag{3}$$

$$x_{ij} \leq x_{jj} \quad i = 1,..n; j = 1,\ldots n \tag{4}$$

where

$n$ is the considered number of demand points
$p$ is the number of facilities or medians
$d_{ij}$ is the distance (cost) between the point $i$ and the facility $j$

$$x_{ij} = \begin{cases} 1 & \text{if the point } i \text{ is assigned to the facility } j \\ 0 & \text{otherwise} \end{cases}$$

$$x_{jj} = \begin{cases} 1 & \text{if the point } j \text{ is a facility} \\ 0 & \text{otherwise.} \end{cases}$$

Restriction (2) prevents a point $i$ from being free, i.e. ensures an associated facility. Restriction (3) establishes the number of facilities (medians). The last condition (4) assures the coherence of the solutions: a demand point $i$ cannot be assigned to a point $j(x_{ij} = 1)$ that is not established as median $(x_{jj} = 0)$.

Besides exact algorithms, several heuristics based on tree search, tabu search and neural networks exist. Some popular ones include tree search (Christofides and Beasley 1982; Bartezzaghi and Colorni 1981), Lagrangian relaxation with branch and bound (Narula, Ogbu and Samuelsson, 1977; Khumawala, 1972; Galvao, 1980; Erlenkotter, 1978; ReVelle and Swain, 1970), tabu search (Ohlemüller, 1997; Rolland *et al*., 1996), and other heuristic and decision techniques (Hribara and Daskin, 1997; Hansen, Mladenovic and Taillard, 1998; Drezner and Guyse, 1999), as well as Kohonen maps (Lozano, Guerrero, Onieva and Larrañeta, 1998). A summary of past results is given in Table 1.

**Table 1** Traditional techniques for the *p*-median problem

| Technique | References |
|---|---|
| Tree Search | Christofides and Beasley (1982) |
| | Bartezzaghi and Colorni (1981) |
| | Narula *et al*. (1977) |
| | Khumawala (1972) |
| Branch and Bound | Galvao (1980) |
| | Erlenkotter (1978) |
| | ReVelle and Swain (1970) |
| Tabu Search | Rolland *et al*. (1996) |
| Kohonen maps | Lozano *et al*. (1998) |
| | Hribara and Daskin (1997) |
| Other heuristics | Hansen *et al*. (1998) |
| | Drezner and Guyse (1999) |

## 3. Modern tools for optimization

This section presents the optimization tools analysed here. We present an initial subsection on EAs, a second one on ANN and a final one with some existing techniques used in the literature for the $p$-median problem.

### 3.1. Evolutionary algorithms. centralised and decentralised models

Genetic algorithms (GAs) were initially developed by Holland in the sixties. They are guided random search algorithms inspired by biological evolution (see e.g., Goldberg, 1989; Holland, 1975). Consequently, the field of *evolutionary computation* (EC), of which genetic algorithms are part, has borrowed much of its terminology from biology. These algorithms rely on the collective learning process within a population of individuals, each of which represents a search point in the space of potential solutions for a given optimization problem (objective function). The population evolves towards increasingly better regions of the search space via probabilistic processes of selection, mutation, and recombination. The selection mechanism has individuals with better objective function value reproduce more often when a new population is formed. Recombination allows for the mixing of parental information that is passed to descendants, and mutation introduces innovation into the population. Usually, the initial population is randomly initialised and the evolution process is stopped after a predefined number of iterations.

Here is an outline of a genetic algorithm from (Khuri and Heitkötter, 1994):

```
Algorithm GA is
        t := 0;
        initialize P(t);
        evaluate P(t);
        while not terminate P(t) do
                t := t + 1;
                P(t) := select P(t − 1);
                recombine P(t);
                mutate P(t);
                evaluate P(t);
        end_while
end GA.
```

In GAs, individuals are generally represented by binary strings that encode the parameters of the problem (genotype). A population is a set of binary vectors, each one being a tentative solution to the problem. Each individual has an associated fitness value computed by the objective function indicating its appropriateness as a problem solution with respect to the rest of the individuals in the population. The genetic operations have a non-deterministic behavior. Recombination is usually performed frequently (high probability) by selecting two individuals, defining one or more random points, and exchanging their contents to create one or two new individuals. Mutation randomly changes the value of one of the positions of an individual and is performed with a small probability. Elitist algorithms in which the current best solution is copied from one generation to the next are very common.

For this paper, we developed and implemented two genetic algorithms: a generational and a cellular genetic algorithm. They were selected as representative of two important subclasses of population-based heuristics: *panmictic* and *structured* algorithms, respectively. In our case, panmictic algorithms consider the entire population as a mating pool for selecting individuals for reproduction, while structured EAs define some kind of neighborhood for each individual, and restrict reproduction to mates selected from it. The reader is referred to (Alba and Tomassini, 2002; Alba and Troya, 1999) for more details on panmictic and structured genetic algorithms.

We now briefly describe our two basic GAs. The genGA, like most GAs described in the literature, is generational, i.e., at each generation, the new population consists entirely of offspring formed by parents in the previous generation (although some of these offspring may be identical to their parents). The pseudocode of a generational GA is as follows:

```
Generational Genetic Algorithm (genGA)
proc Reproductive Cycle (ga):
  for s = 1 to MAX_STEPS do
    p_list = Select(ga.pop);
    for i = 1 to POB_SIZE/2 do
      Crossover(ga.Pc, p_list[i], p_list[i + POB_SIZE/2],
        ind_aux.chrom);
      Mutate(ga.Pm, ind_aux.chrom);
      ind_aux.fitness = ga.Evaluate(Decode(ind_aux.
        chrom));
      Insert_New_Ind(pop_aux, ind_aux);
    end_for
    ga.pop = pop_aux; [elitist}nonelitist]
    Collect_Statistics(ga);
  end_for
end_proc Reproductive Cycle;
```

In each step of this genGA the selection operator gets a set of copies of the best individuals of the population attending to their relative fitness values (i.e., fittest individuals are selected more frequently). From this list, the crossover operator creates new individuals one by one (its last parameter in the pseudocode) by using a given probability of application (first parameter, i.e. crossover is not always applied) to decide whether or not two of the selected parents (second and third arguments) are merged or left unchanged. This merging consists of exchanging the two parts of each parent defined by a randomly selected point of the string encoded in each

solution (using the largest proportion of the best parent). Then, mutation is applied to flip the bit values in the string (not deterministically, but again governed by a low probability of application) to introduce novel information into the population, in order to be able of escaping from local optima. The individual is then evaluated and inserted into a temporary population which is filled by this manner with new individuals to yield the new generation, usually keeping the best individual from the previous generation (elitism) to avoid degradation of the search as it progresses. Finally, the new population replaces the old one (generation).

On the other hand, the population of a cGA is structured in a toroidal 2D grid, and a neighborhood is defined on it to always contain 5 strings (in our case study): the one under consideration plus its north, east, west, and south neighboring strings. The grid used in our tests is a $16 \times 16$ square.

The cGA algorithm is as follows (details are discussed in Alba and Troya (2000)):

```
Cellular Genetic Algorithm (cGA)
proc Reproductive Cycle (ga):
  for s = 1 to MAX_STEPS do
    for x = 1 to WIDTH do
      for y = 1 to HEIGHT do
        n_list = Calculate_neigbors(ga, position(x, y));
        parent1 = Select(n_list);
        parent2 = Select(n_list);
        Crossover (ga.Pc, n_list[parent1], n_list[parent2],
          ind_aux.chrom);
        Mutate(ga.Pm, ind_aux.chrom);
        ind_aux.fitness = ga.Evaluate(Decode(ind_aux.
          chrom));
        Insert_New_Ind(position(x, y), ind_aux,
            [if_better}always], ga, pop_aux);
      end_for
    end_for
    ga.pop = pop_aux;
    Collect_Statistics(ga);
  end_for
end_proc Reproductive Cycle;
```

The operations are mainly the same as those used in the previous genGA, but applied on overlapping subsets of five individuals. Fitness proportional selection is used in the neighborhood along with the mentioned one-point crossover operator. Since a string belongs to several neighborhoods, any change in its contents affects its neighbors in a smooth manner, representing an appropriate tradeoff between a fast convergence and a wide exploration of the search space. Therefore, the cGA also proceeds in generations, and the present population grid is used to create a temporary new grid by repeatedly applying local selection, crossover, and mutation. This temporary grid will later replace the old one in an individual-to-individual manner, affecting only to the locations where the new individuals are better than the existing ones (from a fitness point of view).

*Solving the p-median problem with a GA*

This section begins by discussing the encoding used for representing the problem variables as an integer vector amenable for genetic manipulation. We then will discuss the fitness function used in accordance with the genotype space. In the algorithm, we use bit-flip mutation plus a uniform recombination in which each value from the offspring vector comes from the best of the two parents with a given probability (0.6). Further comments on the operators are given at the beginning of the forthcoming experimental section.

Let us begin with the genotype representation. A very simple mapping is adopted for the $p$-median problem using an integer alphabet. Each individual has exactly $p$ genes, where $p$ is the number of medians, and the allele of each gene represents the index of a facility (median). For instance, consider a 3-median problem with 10 points (potential medians). In our GA, the individual $\langle 3, 5, 8\rangle$ represents a candidate solution (showed in Figure 1) for the problem where points 3, 5 and 8 are selected as medians.

The fitness of an individual is directly related to the previously discussed objective function (1). More specifically, the fitness of an individual ($c$) is given by

$$f(c) = \sum_{i=1}^{n} \left( \min_{1 \le j \le p} \{d(i, c_j)\} \right). \tag{5}$$

In this simple way we match two requirements for the application of metaheuristics: defining the representation of a solution and computing a comparative measure of distance to the solution (i.e., the fitness in our case).

### 3.2. Neural Networks

The field of neural networks (NNs) is an often overlooked source for solutions of the $p$-median problem. One of the most recent and efficient neural models for the $p$-median problem can be found in Domínguez and Muñoz (2002). The proposed NN consists of two layers (allocation layer and location layer) of binary interconnected neurons or processing elements. The authors propose a neural network architecture for the $p$-median problem where two basic approaches are merged.

The first approach formulates a combinatorial optimization problem in terms of minimizing a cost (energy) function for which the global minimum is simultaneously an optimal solution of the $p$-median problem. The simplest approach to

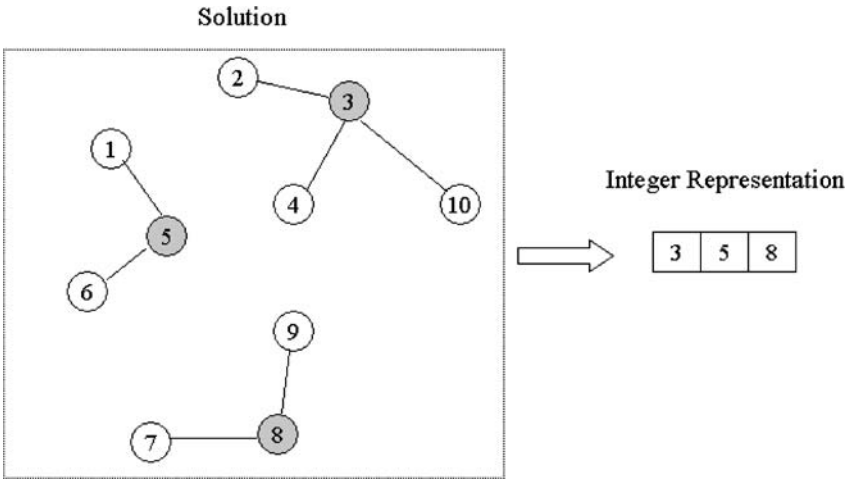


**Fig. 1** Candidate solution for 3-median problem with 10 facilities

construct the energy function ($E$) is to transform the penalty function method. The basic idea in this approach is to transform the constrained problem into an unconstrained one by adding a penalty function terms to the objective function (1). These terms cause a high cost if any constraint is violated, i.e. increasing the objective function by a quantity that depends on the amount by which the constraints are violated.

The second included approach is to design competition-based neural networks where neurons are allowed to compete to become active under certain conditions.

The proposed energy function of the neural network, for which the global minimum is simultaneously the optimum solution of the $p$-median problem, is defined as follows:

$$E = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{q=1}^{p} d_{ij}x_{iq}y_{jq} \tag{6}$$

where

$$x_{iq} = \begin{cases} 1 & \text{if the point } i \text{ is assigned to the cluster } q \\ 0 & \text{otherwise} \end{cases}$$

$$y_{jq} = \begin{cases} 1 & \text{if } j \text{ is a facility in the cluster } q \\ 0 & \text{otherwise.} \end{cases}$$

To guarantee a valid solution, the neural network is organised in disjoint groups as shown in Figure 2. Exactly one neuron per group is activated. That is, the proposed neural network finds solutions to the $p$-median problem minimising the energy function (6) subject to:

$$\sum_{q=1}^{p} x_{iq} = 1 \quad i = 1, \ldots n \tag{7}$$

$$\sum_{j=1}^{n} y_{jq} = 1 \quad q = 1, \ldots p \tag{8}$$

The neural network model based on this formulation guarantees that every stable state of the NN is equivalent to a feasible solution, with the added advantage of removing the tuning phase. This NN implements a gradient descent or hill-climbing search method that only accepts moves (state changes) that improve solution quality, i.e., that decrease the energy. However, this NN should not be viewed as a naive gradient descent technique, but as an ensemble of interconnected processing units with simple computational requirements that can implement complex computation.

The neurons in the same group compete to become active and are updated in parallel. The state of a neuron depends on its activation potential, which is given by the following expressions:

$$h_{x_{iq}} = -\sum_{j=1}^{n} d_{ij}y_{jq} \quad h_{y_{jq}} = -\sum_{i=1}^{n} d_{ij}x_{iq} \tag{9}$$

where $h_{x_{iq}}$ is the activation potential of allocation neuron $x_{iq}$, that is, the negative distance between the demand point $i$ and the median of cluster $q$. In addition, $h_{y_{jq}}$ is the activation potential of the location neuron $y_{jq}$, that is, the total negative distances between the median $j$ and all demand points assigned to cluster $q$.

In each group, the neuron with the maximum activation potential (winner neuron) becomes active. Thus, the computational dynamics is defined as follows:

$$x_{iq}(k+1) = \begin{cases} 1 & \text{if } h_{x_{iq}}(k) = \max\limits_{1\le j\le n}\{h_{x_{jq}}(k)\} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

$$y_{jq}(k+1) = \begin{cases} 1 & \text{if } h_{y_{jq}}(k) = \max\limits_{1\le i\le n}\{h_{y_{iq}}(k)\} \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

The central property of the proposed network is that the computational energy function (6) always decreases (or

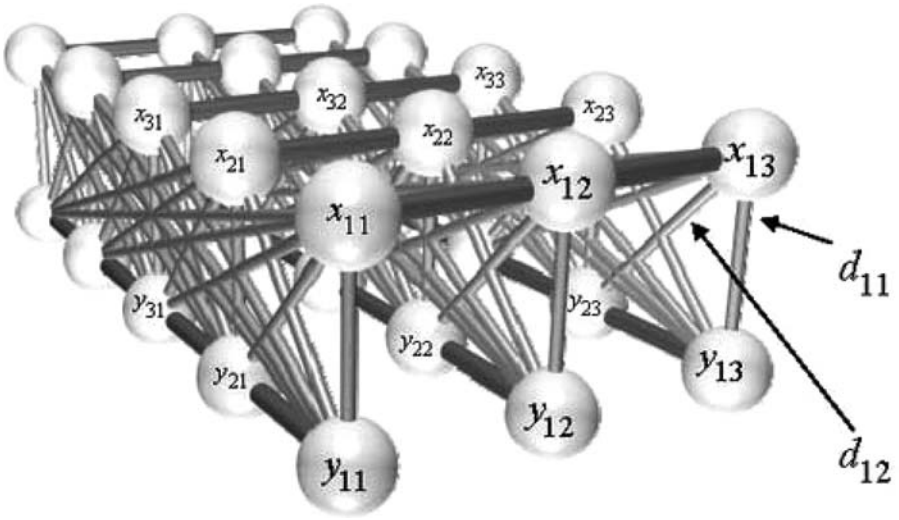


**Fig. 2** Example of a neural arquitecture for the 3-median problem with 5 demand points

remains constant) as the system evolves according to its dynamical rule. Note that the neuron $x_{iq}$ will be activated when the nearest point to the median of group $q$ is $i$, and the neuron $y_{jq}$ will be activated when the point $j$ is the median point of cluster $q$.

To summarise and finalise this section it follows an outline of the proposed neural algorithm (NA) taken from (Domínguez and Muñoz, 2002) to give readers an abstract idea of the internal workings of this technique.

**Algorithm** *NA* ***is***
  $k := 0$;
  *initialize neurons*;
  *evaluate energy* $E(k)$;
  **repeat**
    $k := k + 1$;
    *select a neuron group* $g \in \{1, 2, \ldots, n + p\}$
    *update neurons of the selected group (eqs. 10 or 11)*;
    *evaluate energy* $E(k)$ *(using eq. 6);*
  **until** $(E(k) == E(k - 1))$;
**end** *NA*.

### 3.3. Existing techniques for the *p*-median problem

As pointed out in the introduction, numerous techniques have been proposed for the *p*-median problem. This section briefly summarizes some of the most efficient techniques for the *p*-median problem found in the literature. We just discuss their basic search engine, and let the reader consult the references to learn more of them.

#### *3.3.1. Constructive genetic algorithm*

In the field of evolutionary computation, one of the most recent contributions to solve the *p*-median problem is the *constructive genetic algorithm* (consGA) proposed by Nogueira and Furtado (2001). This is an alternative to the traditional GA approach, particularly in that consGA directly evaluates *schemata*. Schemata are the underlying theoretical hyperplanes being manipulated by a GA according to the well known schema theorem (see e.g., Goldberg, 1989). The population, initially formed only by schemata, is built by directly searching for a population of not only well adapted structures, but also for good schemata.

The consGA works with a dynamic population size that is enlarged after the use of recombination operators, or made smaller as evolution proceeds, guided by an evolution parameter. Tasks to be solved with a consGA are formulated as bi-objective optimization problems. Two fitness functions are defined, one on the space of all possible schemata and a second on actual structures (individuals) that can be obtained using a specific representation (see Nogueira and Furtado, 2001, for details and results).

#### *3.3.2. Scatter search*

An additional metaheuristic recently proposed for the *p*-median problem is the replicated parallel scatter search (RPSS) proposed by García-López *et al.* (2003). Scatter Search (SS) is a population-based metaheuristic that constructs solutions by combining others from a set of solutions, named the reference set. The method generates a reference set from a population of solutions. Then a subset is selected from this reference set. The selected solutions are combined to get starting solutions to run a local search procedure. The resulting improvement can motivate the updating of the reference set and even the updating of the population of solutions.

In the cited paper, the authors analyse three different parallelization strategies for the SS algorithm. The first two strategies run the same algorithm by following two different parallel strategies only aimed at reducing the running time of

the search of the optimum. The first strategy runs in parallel every local search step, while the second one performs the reference set operations and improvement in parallel. The last strategy increases the exploration capabilities of SS (i.e., it is a new algorithm) by running the SS in parallel on separated populations (RPSS). Authors test these three algorithms with the FL1400 instance of the *p*-median problem obtained from the TSPLIB (Reinelt, 1991). The most accurate results are found with the last parallelization scheme: the replicated parallel scatter search (RPSS), which resembles the well known multipopulation island models in other EAs.

## 4. Computational results

We now perform a computational analysis to evaluate the performance of the two proposed GAs (generational genGA, and cellular cGA), the neural model (NA), the constructive genetic algorithm (consGA), and the parallel scatter search algorithm (RPSS). We are interested in comparing the final accuracy of the provided solutions of our methods, as well as in comparing the effort needed to locate such solutions. Such an efficiency measure is attained by computing the number of visited points, as an indicator of the effort needed.

Initially, it is common sense to expect customized solvers to have some advantage, and in this case we are interested in quantifying that advantage, especially in relation to the effort of algorithm construction, knowledge re-use, and efficiency.

We performed 50 independent runs of each algorithm to provide meaningful average values in the tables. We explicitly show the error of the best solution as a percentage of its distance to the known optimum value for each instance. The parameters used in the generational/cellular GAs are shown in Table 2.

We have organised the results in two subsections. First, we address medium size instances of up to 400 points and 133 medians. We also consider a real world problem based on the Australian postal network. This problem also fits into the medium-size category, since it consists of 200 points and up to 100 medians. The next subsection encompasses a set of experiments on large problem instances, dealing with a number of points between 500 and 900, plus a still larger problem of 1400 points. Still larger values are deferred for detailed study in an ongoing project.

**Table 2** GA parameters

| Operator | Probability |
|---|---|
| Bit-Flip mutation | 0.2 |
| Uniform crossover | 1.0 (60% best parent) |
| Roulette wheel selection | |
| $16 \times 16 = 256$ individuals | |

### 4.1. Medium size and real-world instances

We first analyse the results of the constructive GA, the neural algorithm and the generational GA. In Table 3 we can see that the consGA is quite accurate, and that our genGA is similarly accurate for some of the data sets. The exception is for the largest values of medians, for which consGA is more efficient than genGA. However, the original work did not provide the performance of such algorithm for more than 300 points and more than 10 medians, while our genGA showed results in this case and even for larger instances. The NA error is in the range of genGA in most cases, although in the presence of a large number of medians its accuracy clearly improves over the genGA (as also occurs in the cGA). In most cases, the cGA provides a smaller error than the genGA. It is difficult to predict what would have occurred with consGA for more than 300 points, but it seems to be good at managing a large number of medians with respect to our GAs.

Since we want to evaluate our two most recent solvers (the cGA and the NA) on a real world problem for future comparisons, we now present the distance of the worked solution

**Table 3** Percentage error comparison

| Problems | | Percentage Errors (%) | | | |
|---|---|---|---|---|---|
| Name | Size (*n*,*p*) | consGA | NA | genGA | cGA |
| pmedl | (100, 5) | 0.00 | 0.28 | 0.00 | 0.00 |
| pmed2 | (100,10) | 0.00 | 1.73 | 0.00 | 0.29 |
| pmed3 | (100,10) | 0.00 | 0.33 | 0.00 | 0.00 |
| pmed4 | (100,20) | 0.00 | 2.99 | 4.85 | 3.00 |
| pmed5 | (100,33) | 0.36 | 23.03 | 22.73 | 12.18 |
| pmed6 | (200, 5) | 0.00 | 0.18 | 0.00 | 0.00 |
| pmed7 | (200,10) | 0.00 | 1.05 | 0.00 | 0.44 |
| pmed8 | (200,20) | 0.20 | 3.82 | 5.31 | 5.13 |
| pmed9 | (200,40) | 0.73 | 7.53 | 19.24 | 13.50 |
| pmed10 | (200,67) | 0.15 | 29.54 | 44.14 | 44.62 |
| pmed11 | (300, 5) | 0.00 | 0.32 | 0.00 | 0.08 |
| pmed12 | (300,10) | 0.04 | 0.57 | 0.09 | 1.10 |
| pmed13 | (300,30) | n/a | 3.29 | 3.96 | 6.10 |
| pmed14 | (300,60) | n/a | 8.77 | 25.40 | 22.41 |
| pmed15 | (300,100) | n/a | 26.95 | 45.58 | 44.13 |
| pmed16 | (400, 5) | n/a | 1.00 | 0.15 | 0.00 |
| pmed17 | (400,10) | n/a | 3.64 | 1.99 | 0.00 |
| pmed18 | (400,40) | n/a | 9.67 | 6.97 | 13.45 |
| pmed19 | (400,80) | n/a | 20.77 | 27.14 | 28.72 |
| pmed20 | (400,133) | n/a | 47.18 | 56.51 | 57.24 |

**Table 4** Results for Australian postal data (200 points)

| | Fitness ($10^{-6}$) | |
|---|---|---|
| Medians ($p$) | cGA | NA |
| 10 | 0.99 | 0.99 |
| 20 | 1.49 | 1.41 |
| 30 | 1.91 | 1.87 |
| 40 | 2.21 | 2.26 |
| 50 | 2.54 | 2.62 |
| 60 | 2.82 | 2.98 |
| 70 | 3.19 | 3.33 |
| 80 | 3.53 | 3.71 |
| 90 | 3.86 | 4.07 |
| 100 | 4.25 | 4.46 |

and its corresponding fitness for the Australian postal problem. This problem comprises 200 points and from 10 to 100 medians. Table 4 shows that the cellular GA achieves a really small fitness value, which makes us think that the computed solution is really near the optimum. Unfortunately, no other results were found for this benchmark and the $p$-medians problem (although some results do exist with the same data but for the $p$-hub problem).

One interesting feature of this problem is that we can plot the result graphically in connection with actual cities in Australia. This again has never be addressed in the past, and so we just guess by following an intuitive best effort study of data and the Australian geography. Of course, the city assignments used in the result of Figure 3 is not claimed to be completely correct since the original numerical data were not labelled with this information.

### 4.2. Large size instances

This subsection encompasses a comparison of the cellular GA, the generational GA and the NA when faced with problems with a high number of points and medians. Table 5 shows the numerical accuracy of the algorithms. We can see that the cellular approach offers a structured evolution that gives a final error smaller than the panmictic generational GA, as with the smaller sized instances. In some cases, the two algorithms (cGA and GA) report a similar error. Although for most cases the cGA is more accurate than genGA (15.81 versus 15.97), the NA is the most precise (12.31) with respect to the average error. These instances are in the range of 500 to 900 points, which represents a considerable step up in size compared to other existing works in literature.

We finally perform a comparison test between the cellular GA with respect to a parallel distributed scatter search previously presented in Section 3.3.2. We consider the FL1400 data set taken from the TSPLib data set for the TSP problem. This data accounts for 1400 points, and we analyse 10 cases, for $p$ from 10 to 100 in steps of 10.

Table 6 shows that the final accuracy is quite similar for the cellular GA and the RPSS algorithm, with the latter having a slight advantage. This table reports two flavors of the cGA with different disposition of the individuals in the grid ($16 \times 16$ and $32 \times 32$), since we have some experiences in improving the accuracy through changes in the shape of the population (Alba and Troya, 2000). Although our algorithm

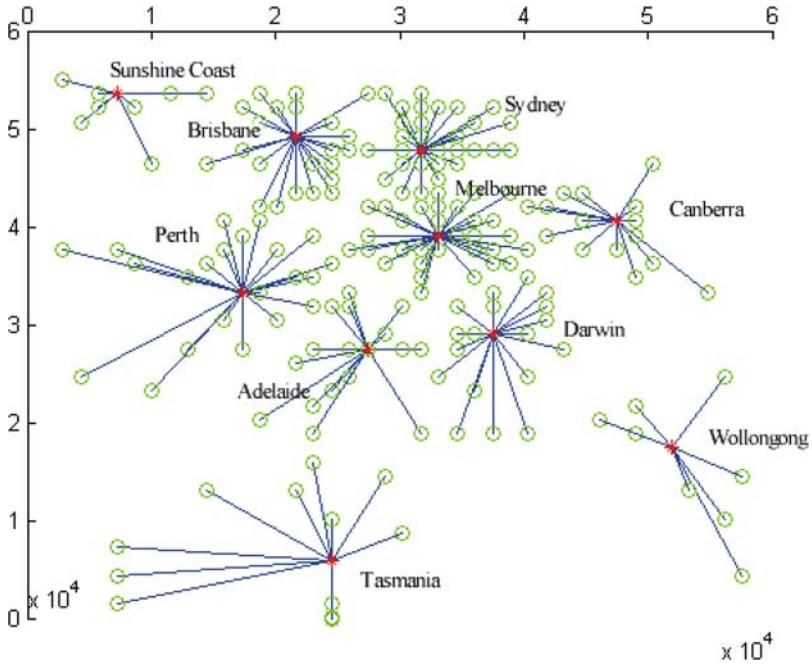


**Fig. 3** Best 10-median solution found for the Australian postal problem

**Table 5** Genetic and neural algorithms (large size problems)

| Problems | | Percentage Error (%) | | |
|---|---|---|---|---|
| Name | Size (*n*,*p*) | cGA | genGA | NA |
| pmed21 | (500, 5) | 0.00 | 0.00 | 1.71 |
| pmed22 | (500,10) | 2.05 | 0.00 | 2.51 |
| pmed23 | (500,50) | 13.92 | 15.50 | 9.46 |
| pmed24 | (500,100) | 31.00 | 31.81 | 20.67 |
| pmed25 | (500,167) | 61.27 | 59.30 | 51.75 |
| pmed26 | (600, 5) | 0.07 | 0.00 | 2.43 |
| pmed27 | (600,10) | 1.36 | 0.04 | 4.89 |
| pmed28 | (600,60) | 14.21 | 18.32 | 11.07 |
| pmed29 | (600,120) | 33.56 | 33.60 | 21.07 |
| pmed30 | (600,200) | 57.97 | 60.53 | 46.56 |
| pmed31 | (700, 5) | 0.00 | 0.00 | 0.55 |
| pmed32 | (700,10) | 0.52 | 0.04 | 5.10 |
| pmed33 | (700,70) | 18.26 | 18.83 | 12.64 |
| pmed34 | (700,140) | 37:37 | 38.43 | 24.79 |
| pmed35 | (800, 5) | 0.64 | 0.00 | 0.00 |
| pmed36 | (800,10) | 0.67 | 0.00 | 2.57 |
| pmed37 | (800,80) | 21.49 | 22.17 | 10.22 |
| pmed38 | (900, 5) | 0.69 | 0.00 | 2.21 |
| pmed39 | (900,10) | 0.69 | 0.06 | 3.39 |
| pmed40 | (900,90) | 20.53 | 20.85 | 12.56 |
| Average Error | | 15.81 | 15.97 | 12.31 |

**Table 6** cGA and RPSS comparison for TSP - FL1400

| Medians ($p$) | cGA (grid $16 \times 16$) | cGA (grid $32 \times 32$) | RPSS |
|---|---|---|---|
| 10 | 101526.73 | 101437.58 | 101249.47 |
| 20 | 58216.96 | 58228.47 | 57857.55 |
| 30 | 44716.07 | 44609.72 | 44013.02 |
| 40 | 35645.16 | 35662.51 | 35002.02 |
| 50 | 29923.27 | 29847.77 | 29089.71 |
| 60 | 26143.08 | 26030.17 | 25160.40 |
| 70 | 23196.93 | 23210.25 | 22125.46 |
| 80 | 20799.90 | 20794.05 | 19870.29 |
| 90 | 18830.49 | 18829.11 | 17987.91 |
| 100 | 17287.01 | 17352.31 | 16552.48 |

does not boast clear superiority, we consider such results extremely interesting, since we achieved them with a "simple" non parallel GA, which contrasts with the high customization and implementation effort for running a parallel algorithm like RPSS. In fact, this is a major achievement of our work: the same algorithm (cGA) is able of outperforming a set of existing algorithms, without local search, parallelism and even without sophisticated parameter tuning. This means that we can still easily extend the algorithm with such features to outperform the rest, which clearly indicates a future research step.

## 5. Conclusions

This work analysed a structured cellular GA, a canonical generational GA, and a neural model; we compared these against each other and against several other algorithms found in literature like the constructive GA and the parallel scatter search. We conclude that both the cellular GA and the neural optimiser have their own advantages. The cGA is accurate, fast, simple to program and open to new advances in specialized operators (local search, new representations, etc.). On the other hand, NA is still more accurate, but represents a customized model of narrower applicability. The effort devoted to cGA can be profitable in similar problems and open to future improvements appearing in similar algorithms, while the neural model is very accurate at a price of larger computational times, difficult generalisation, and higher programming effort.

Here, we have considered problems with a large number of points and medians in the aim of providing a wide set of results for future extensions. Since local search and parallelism have led to accurate algorithms in the past for the $p$-median problem, we intend to include them in our algorithm, yielding new and improved methods for the optimization of such NP-hard tasks, similar to those models reported in Alba and Troya (1999).

The fact that the same algorithm is able of computing that accurate solutions along with its simplicity can be a step towards making the research community of the potential inside structured population-based heuristics. This is an important point, since most authors pay credit to algorithms reporting high accuracy results but that are never actually used, since they are difficult to implement or understand (as occurs with parallelism and sophisticated local search in many research communities). The cellular GA is easy to implement and, at the same time, accurate, which follows the "keep it simple" approach that is behind most well known metaheuristics.

**Acknowledgements** This work has been partially funded by the Spanish Ministry of Science and Technology and FEDER under contract TIC2002-04498-C05-02 (the TRACER project).

## References

Alba E. and Tomassini M. 2002. Parallelism and Evolutionary Algorithms. IEEE Transactions on Evolutionary Computation 6(5): 443–462.

Alba E. and Troya J. M. 1999. A Survey of Parallel Distributed Genetic Algorithms. Complexity 4(4): 31–52.

Alba E. and Troya J. M. 2000. Cellular Evolutionary Algorithms: Evaluating the Influence of Ratio. In: M. S. *et al*. (ed.). Parallel Problem Solving from Nature, PPSN VI, Vol. 1917 of Lecture Notes in Computer Science. pp. 29–38.

Bartezzaghi E. and Colorni A. 1981. A search tree algorithm for plant location problems. European Journal of Operational Research 7: 371–379.

Christofides N. and Beasley J. 1982. A tree search algorithm for the problem of *p*-medians. European Journal of Operational Research 10: 196–204.

Domínguez E. and Muñoz J. 2002. An efficient neural network for the *p*-median problem. Lecture Notes in Artificial Intelligence 2527: 460–469.

Drezner Z. and Guyse J. 1999. Application of decision analysis to the Weber facility location problem. European Journal of Operational Research 116: 69–79.

Erlenkotter D. 1978. A dual-bounded procedure for uncapacitated facility location. Operations Research 26(6): 992–1009.

Galvao R. D. 1980. A dual-bounded algorithm for the problem of *p*-medians. Operations Research 28: 1112–1121.

García-López F., Melián-Batista B., Moreno-Pérez J. A., and Moreno-Vega J. M. 2003. Parallelization of the Scatter Search for the *p*-median Problem. Parallel Computing 29: 575–589.

Garey M. and Johnson D. 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. San Francisco, CA: W. H. Freeman and Co.

Goldberg D. E. 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley.

Hansen P., Mladenovic N. and Taillard E. 1998. Heuristic solution of the multisource Weber problem ace to problem of *p*-medians. Operations Research Letters 22: 55–62.

Holland J. H. 1975. Adaptation in Natural and Artificial Systems. Ann Arbor, Michigan: The University of Michigan Press.

Hribara M. and Daskin M. S. 1997. A dynamic programming heuristic for the problem of *p*-medians. European Journal of Operational Research 101: 499–508.

Kariv O. and Hakimi S. 1979. An Algorithmic Approach to Network Location Problem. Part 2: The *p*-Median. SIAM J. Appl. Math. 37: 539–560.

Khumawala B. M. 1972. An efficient branch and bound algorithm for the warehouse location problem. Management Science 18(12): 718–731.

Khuri S., Bäck T. and Heitkötter J. 1994. An Evolutionary Approach to Combinatorial Optimization Problems. In: Proceedings of the 22nd ACM Computer Science Conference. Phoenix, Arizona, pp. 66–73.

Lozano S., Guerrero F., Onieva L. and Larrañeta J. 1998. Kohonen maps for solving to class of location-allocation problems. European Journal of Operational Research 108: 106–117.

Manderick B. and Spiessens P. 1989. Fine-grained parallel genetic algorithms. In: Schaffer J. D. (ed.), Proceedings of the Third International Conference on Genetic Algorithms. pp. 428–433.

Narula S. C., Ogbu U. I. and Samuelsson H. M. 1977. An algorithm for the problem of *p*-medians. Operations Research 25: 709–713.

Nogueira L. A. and Furtado J. C. 2001. Constructive genetic algorithm for clustering problems. Evolutionary Computation 9(3): 309–328.

Ohlemüller M. 1997. Taboo search for large location-allocation problems. Journal of the Operational Research Society 48: 745–750.

Reinelt G. 1991. TS*P*-Lib a travelling salesman library. ORSA Journal of Computing 3: 376–384. http://www.iwr.uniheidelberg.de/groups/comopt/software/TSPLIB95/.

ReVelle C. and Swain R. 1970. Central facilities location. Geographical Analysis 2: 30–42.

Rolland E., Shilling D. A., and Current J. R. 1996. An efficient tabu search procedure for the *p*-median problem. European Journal of Operational Research 96: 329–342.

Syswerda G. 1991. A Study of Reproduction in Generational and Steady-State Genetic Algorithms. In: Rawlins G. J. E. (ed.), Foundations of Genetic Algorithms. pp. 94–101.