Engineering Optimization
Vol. 39, No. 7, October 2007, 831–855

Evolutionary algorithms applied to reliable communication network design

SERGIO NESMACHNOW*†, HÉCTOR CANCELA† and ENRIQUE ALBA‡

†Facultad de Ingeniería, Universidad de la República, Herrera y Reissig 565, Montevideo, Uruguay
‡Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga,
Campus Teatinos, Málaga, Spain

(Received 8 May 2006; in final form 5 January 2007)

Several evolutionary algorithms (EAs) applied to a wide class of communication network design problems modelled under the generalized Steiner problem (GSP) are evaluated. In order to provide a fault-tolerant design, a solution to this problem consists of a preset number of independent paths linking each pair of potentially communicating terminal nodes. This usually requires considering intermediate non-terminal nodes (Steiner nodes), which are used to ensure path redundancy, while trying to minimize the overall cost. The GSP is an NP-hard problem for which few algorithms have been proposed. This article presents a comparative study of pure and hybrid EAs applied to the GSP, codified over MALLBA, a general purpose library for combinatorial optimization. The algorithms were tested on several GSPs, and asset efficient numerical results are reported for both serial and distributed models of the evaluated algorithms.

Keywords: Evolutionary algorithms; Network design; Generalized Steiner problem

1. Introduction

A major problem in the design of a communication network is the construction of a connection topology which guarantees some reliability properties. Network reliability measures the probability of success in establishing communication between a pair of nodes, which is an important factor in the quality of the service offered to users. The evaluation of exact network reliability is NP-hard (Ball 1979); an alternative approach is to use vulnerability parameters, which are related to reliability but are easier to evaluate. One important parameter is the number of different paths allowing communication between any pair of terminal nodes. This requirement is in general not uniform for every pair of nodes, and thus different pairs could require a different minimum number of paths between them.

The rapid development of network infrastructures, software, and internet services has been driven by the growing demand for data communications over the last 20 years. This is the reason for a renewed interest in network design problems, including optimal allocation of

*Corresponding author. Email: sergion@fing.edu.uy

Engineering Optimization
ISSN 0305-215X print/ISSN 1029-0273 online © 2007 Taylor & Francis
http://www.tandf.co.uk/journals
DOI: 10.1080/03052150701503553

antennas, frequency assignment to cellular phones, and structural design problems relating to routing information through the net (Corne *et al.* 2000, Pedrycz and Vasilakos 2001). Since the size of existing communication networks is continuously increasing, the underlying instances of related optimization problems frequently pose a challenge to existing algorithms. In consequence, the research community is currently searching for new algorithms able to replace and improve the traditional exact ones, whose low efficiency often makes them useless for solving real-life problems of large size in a reasonable time.

In this sense, heuristic algorithms have been applied to reliable network design problems. Although they can sometimes fail in computing a true optimum for the problem, they provide appropriate quasi-optimal solutions which satisfy network designers. Among a whole new set of heuristics and modern optimization techniques, evolutionary algorithms (EAs) have emerged as flexible and robust methods for solving the underlying complex optimization problems found in network design, and have also been applied in many other areas of application such as industry, mathematics, economy, telecommunications, and bioinformatics (Bäck *et al.* 1997).

This work focuses on a wide class of problems that can be modelled under the *generalized Steiner problem* (GSP). Given a communication network with some distinct nodes, called *terminal nodes*, the GSP consists of designing a minimum cost subnetwork verifying a set of prefixed minimum connection requirements for pairs of terminal nodes. Usually, the minimization of the connection costs conflicts with the maximization of the reliability of the resulting network. For example, a minimum cost model that does not account for additional restrictions (*e.g.* a minimum degree of path connectivity) would lead to a tree-like network topology. Solutions of this type with no path redundancy cannot be used in real-life scenarios, as they do not tolerate failure of even a single component.

Solving a GSP on a communication network does incorporate the additional requirements over terminal node connectivity that real-life situations demand, hence guaranteeing high network reliability. The GSP has rarely been addressed in the past; this work presents the development of some pure and hybrid EAs with the aim of solving the GSP with high numerical accuracy and efficiency.

The article is structured as follows. The GSP, its mathematical formulation, and popular variants are presented in the next section. Section 3 contains an overview of previous work related to evolutionary techniques applied to solve simplifications of the GSP. Section 4 describes the algorithms used in this work and their most salient features. Section 5 presents MALLBA, the public C++ algorithmic environment on which algorithms were implemented. Implementation details are given in section 6, and the experiments and results are discussed in section 7. Conclusions and future work are presented in section 8.

2. Generalized Steiner problem

This section includes a formal characterization of the GSP and shows a case study which exemplifies the intrinsic difficulty of the problem. Then, a general mathematical formulation is discussed, together with some popular variants of the general problem specification.

2.1 *Mathematical formulation*

The following GSP formulation is based on the definition for the *minimum generalized Steiner network* problem, included in the compendium of NP optimization problems by Kahn and Crescenzi (2006).

Consider the following elements.

- An undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges representing the bidirectional (full-duplex) communication channels.
- A cost matrix C associated with the edges of the graph G.
- A fixed subset of the node set $T \subseteq V$, called the terminal node set, whose cardinality is $n_T = |T|$ such that $2 \leq n_T \leq n$, where $n = |V|$ is the cardinality of the node set V.
- An $n_T \times n_T$ symmetric matrix $R = r_{ij}, i, j \in T$, whose elements are non-negative integers indicating the connectivity requirements (the number of disjoint paths required) between any pair of terminal nodes (i, j).

Solving the GSP consists of finding a minimum cost subgraph G_T in G, where any pair of nodes $i, j \in T$ is r_{ij} edge-connected in G_T. This last condition means that there must exist r_{ij} *disjoint paths* with no single shared edge between terminal nodes i and j. The nodes not belonging to the terminal node set are known as Steiner nodes. No connection requirements are formulated over them, and they can be either included or omitted in the optimum solution, depending on the convenience of their use.

The previous description corresponds to the edge-connectivity GSP version, which is useful when edges are subject to failures but nodes are assumed never to fail. An analogous formulation can be used when nodes can fail; in this case, the problem specification demands the existence of node-disjoint paths between terminal nodes.

2.2 *A GSP example*

Consider the graph G shown in figure 1, where terminal nodes are dark grey and labelled A–E, and Steiner nodes are light grey. The costs associated with each edge are also specified, and the connection requirements are defined in table 1.

One minimum cost solution of the GSP example in figure 1 is shown in figure 2 (other optimum solutions could exist). In order to connect the five terminal nodes and, at the same time to satisfy the connection requirements, the solution includes eight of the 10 Steiner nodes, geographically selected as the optimum in terms of number and location. The solution graph includes only 16 of the original 26 edges, diminishing the original cost from 71 to 36.

Figure 3 shows a real-life network scenario, which illustrates the benefit of having multiple alternative communication paths in order to guarantee the reliability of communication

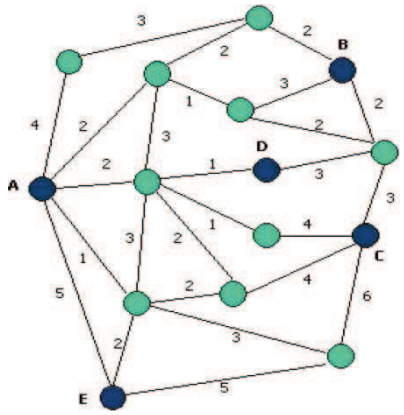

Figure 1. Original graph G for a GSP example (cost 71).

Table 1. Connection requirements for the GSP example shown in figure 1.

Node	Node	No. of required paths
A	B	3
A	C	3
B	D	2
B	C	3
A	E	1
A	D	2

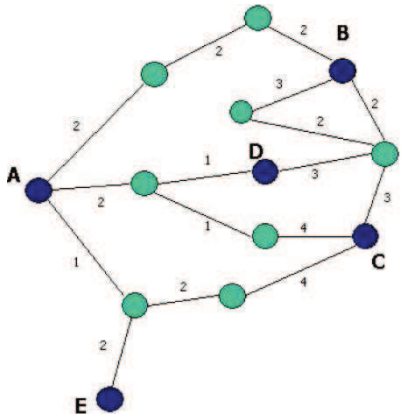

Figure 2. GSP solution for graph G in figure 1 (cost 36).

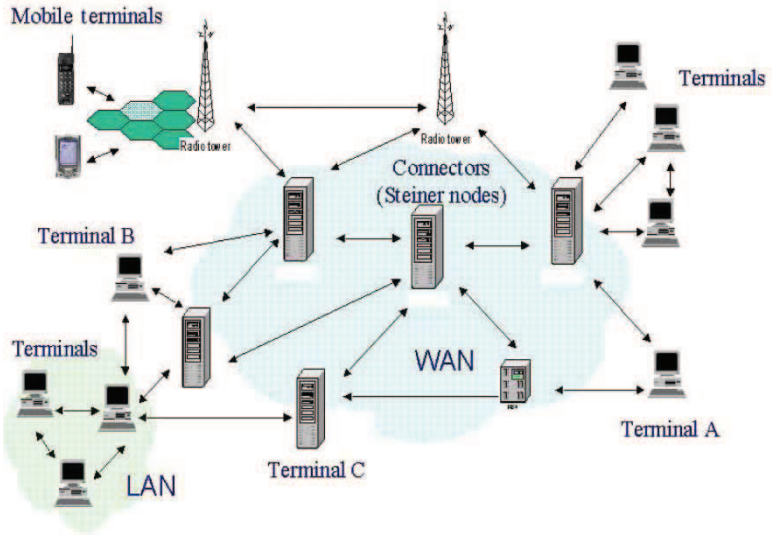

Figure 3. Real-life scenario.

between terminals. For example, there are two disjoint paths between terminal nodes A and B, and three disjoint paths between terminal nodes B and C, showing a robust and fault-tolerant design. Obviously, this case does not reflect a minimum cost scenario, since it contains numerous superfluous links, and it is possible to optimize the network design.

2.3 *Mathematical model for optimization*

A mathematical model can be defined for the edge-connectivity GSP based on assigning more than one variable to each edge $(i, j) \in E$. The binary variable x_{ij} indicates the presence (or absence) of the edge (i, j) in the solution, and the (real-valued) variables y_{ij}^{kl} indicate the amount of commodity to be sent along a path connecting terminal nodes k and l through the edge (i, j) in the direction from i to j, *i.e.* the variable y_{ij}^{kl} takes a value greater than 0 if edge (i, j) is used in some path connecting k and l, and 0 otherwise.

The GSP solutions are given by the optimum solutions of the following integer linear programming problem (GSP-ILP), originally presented by Winter (1987):

$$\text{Min} \sum_{(i,j)\in E} c_{ij} x_{ij} \tag{1}$$

subject to

$$x_{ij} \geq y_{ij}^{kl} + y_{ji}^{kl}; \quad \forall (i, j) \in E; \quad \forall k, l \in T; \; k \neq l \tag{2}$$

$$\sum_{(k,j)\in E} y_{kj}^{kl} \geq r_{kl}; \quad \forall k, l \in T; \; k \neq l \tag{3}$$

$$\sum_{(i,j)\in E} y_{il}^{kl} \geq r_{kl}; \quad \forall k, l \in T; k \neq l \tag{4}$$

$$\sum_{(p,j)\in E} y_{pj}^{kl} - \sum_{(i,p)\in E} y_{ip}^{kl} \geq 0; \quad \forall k, l \in T; \quad \forall p \in V \backslash \{k, l\} \tag{5}$$

$$x_{ij} \in \{0, 1\}; \quad \forall (i, j) \in E \tag{6}$$

$$0 \leq y_{ij}^{kl} \leq 1 \quad \forall i, j : (i, j) \in E; \quad \forall k, l \in T; \; k \neq l. \tag{7}$$

In this model, expression (1) gives the objective function of the problem, which corresponds to minimizing the sum of the costs of the edges present in the solution. Restriction (2) couples the binary variables x_{ij} with the real valued variables y_{ij}^{kl} and y_{ji}^{kl}, by stating that if the edge (i, j) has been used for at least some pair of terminals (k, l), then it must be present in the final solution. Restrictions (3) and (4) ensure that the flow from k to l is at least r_{kl}, the connectivity requirements for the problem. Restriction (5) corresponds to classical flow conservation equations, and restrictions (6) and (7) give the integrality non-negativity constraints.

If $U = \{u_{ij}\}$ are the values of the x_{ij} variables in an optimal solution of GSP-ILP, the set of edges given by $\{(i, j) \in E / u_{ij} = 1\}$ defines a solution subgraph G_{SOL} (Robledo 2001).

2.4 *Variants and complexity of Steiner problems*

The GSP complexity is a consequence of its general formulation, since the problem imposes non-uniform connection requirements among terminal nodes. Some variants of the Steiner problem can be obtained by simplifying this requirement. The problem subclass known as the *k-connection problems* demand a common number of disjoint paths (k) for every pair of terminal nodes. The simplest case of a Steiner problem requires that a single path exist between terminal nodes. Any optimal solution to this problem exhibits a tree topology, and thus the problem is called the *Steiner tree problem.*

The GSP is in the NP-hard problem class (Kahn and Crescenzi 2006). Even the Steiner tree problem, with the less general path restriction requirement, is NP-complete (Karp 1972). As a consequence, all these related problem classes are not amenable to exact methods as the size

of problems grows. Instead, heuristic methods are usually employed since they can produce acceptable or optimal solutions in reasonable times.

3. Related work: evolutionary techniques applied to the Steiner problem class

This section presents a survey on the existing attempts to solve a GSP-like problem by using evolutionary techniques. After such a study of the state of art for Steiner problems, the conclusion drawn is that the simpler variants have often been addressed, but the EA and heuristics communities have mainly ignored the general problem class. Although the specific EAs used in this work will be explained in a later section, a brief introduction is offered here for readers unfamiliar with the terminology.

3.1 *Evolutionary algorithms*

EAs are stochastic search methods that have been successfully applied in many real applications of high complexity. An EA is an iterative technique which applies stochastic operators to a pool of individuals (the population) in order to improve their *fitness*, a measure related to the objective function. Every individual in the population is the encoded version of a tentative solution. Initially, this population is randomly generated. An evaluation function associates a fitness value with every individual, indicating its suitability for the problem. Iteratively, the application of operations such as the recombination of parts of two individuals (cross-overs) or random changes in their contents (mutations) is guided to tentative solutions of higher quality by a selection-of-the-best technique. A particularly popular type of EA is the genetic algorithm (GA), in which all the operators described above are included.

3.2 *Application of EAs to Steiner problems*

Hesser *et al.* (1989) made the first GA proposal for a Steiner tree optimization. They used a codification containing spatial information for Steiner points and a fitness function evaluating the associated Steiner tree cost. The results were not conclusive, because the authors did not find significant differences when comparing the GA with a highly specialized heuristic for finding Steiner trees.

Kapsalis *et al.* (1993) presented a GA for the Steiner tree problem using node-based encoding and working on a population including infeasible solutions. Their GA integrated a penalty function in the fitness evaluation. The authors reported that GA techniques provide a successful method for finding solutions to the Steiner tree problem in sparse graphs.

David *et al.* (1993) described a steady-state GA applied to the design of survivable networks. The GA was used to assign link capacities, subject to network routing demands and survivability constraints, with the objective of minimizing the total network link cost. This proposal showed good results for high dimensional problems when compared with both a greedy heuristic and an integer programming technique.

At the same time, Julstrom (1993, 1994) studied the rectilinear Steiner problem. In this problem the underlying graph is planar, since it is assumed that terminal nodes are located in the Euclidean plane. Additionally, the connections between nodes must be horizontal or vertical. This problem has a direct application to network design problems, mechanical circuit systems, and VLSI packet design. Julstrom proposed a mixed representation (binary and non-binary), mapping spanning trees onto the string space. He introduced a modified cross-over operator

to avoid 'excessive' information loss when generating offsprings. Later, Julstrom (2001) presented a hybrid algorithm which incorporated a specific heuristic to initialize the population, reporting better-quality results and improving the overall performance.

Esbensen (1994) suggested alternatives to the work of Kapsalis *et al.* (1993), extending their ideas to solve the Steiner tree problem. Using a binary codification and a decoding procedure based on distance network heuristic, he introduced a significantly different approach, proposing to work only with feasible solutions and thus avoiding penalty terms in the fitness function. In addition, Esbensen noticed that, by using such a codification, different genotypes could exist for a single Steiner tree. In order to improve the performance of the cross-over operator and to make it independent of equivalent genotypic representations, he also included an inversion operator which reorders the tuples of a given genotype without changing its phenotypic expression. As an extension of this work, Esbensen and Mazumder (1994) presented a new GA for a Steiner tree problem applied to VLSI circuits design, which showed higher-quality results and performance than two other well-known heuristics for the same problem.

From a different point of view, several recent papers have addressed network design problems and searched for Steiner trees by applying ideas of existing evolutive techniques.

Zhu *et al.* (1998) implemented a hybrid algorithm to find Steiner trees, combining Esbensen's ideas and their own proposals for improving the multipoint routing requirement optimization over the network.

Hwang *et al.* (2000) developed a GA for the Steiner tree problem and applied it to a multicast routing problem. In this work, each chromosome codifies paths from a source node to a set of destination nodes, using the codification schema previously proposed by Hiramatsu *et al.* (1993). The performance of the GA-based multicast routing algorithm is evaluated in terms of the cost of the multicast tree found. The authors reported that their GA obtained better results than a well-known heuristic for finding Steiner trees.

With respect to the same multicast routing problem, Xianwei *et al.* (2000) proposed a hybrid GA for finding a connection topology which guarantees the 'best use' of network resources under specified constraints. Using a binary representation to encode actual Steiner nodes, the GA combines a heuristic to decode the genotype and build the associated Steiner tree. This codification scheme ensures working with feasible solutions, avoiding penalty terms in the fitness measure. Also, standard network reduction techniques and an evolution strategy have been used to reduce the total cost. The authors did not provide numerical results, but they claimed to reach better-quality solutions than those obtained with deterministic heuristics, even within moderate execution times.

Ljubic *et al.* (2000) presented a hybrid GA applied to the edge-biconnectivity augmentation problem, which combined a reduction technique and a GA using binary representation for candidate edges to be added to the original graph. Two strategies were presented to deal with infeasible solutions during the evolutive process: the simplest variant detects and discards infeasible solutions, while the second method proposes a repairing mechanism for infeasible solutions, adding low-cost edges until the graph becomes edge-biconnected. The GA itself was adapted from the work of Ljubic and Kratica (2000) to solve the node-biconnectivity augmentation problem.

Galiasso and Wainwright (2001) studied the multipoint routing problem, extending the hybrid GA proposed by Zhu *et al.* (1998). They used Esbensen's ideas to find Steiner trees. In addition, the GA was able to determine the optimum ordering for processing requirements and to optimize the bandwidth percentage assigned to network paths. The authors reported better results than those obtained by Zhu *et al.* (1998).

In the VLSI circuit application domain, Wakabayashi (2002) presented a GA for finding minimum rectilinear Steiner trees. In this work, the chromosome only codifies the topological information about the Steiner tree recursively (parent–children relationship). The Steiner tree

is constructed using information taken from the fitness evaluation process, applying a method based on the traditional Kruskal algorithm to find minimum spanning trees over graphs. The fitness function evaluates the total network cost and the maximum delay between source and destination. The GA uses a special cross-over operator, called subtree exchange, which claims to be effective in terms of quality.

None of the authors cited above tackled the GSP directly. Rather, they solved simplified problem examples such as the Steiner tree problem or k-connection problems.

Recently, Arraga *et al.* (2003) have developed a specific GA to solve the GSP, trying to keep the codification and variation operators as simple as possible. The algorithms presented have used the same problem codification, but include additional different EAs.

4. Algorithms used in the study

This section describes the optimization EAs used in the study: GA, in both the canonical formulation and the CHC variant, simulated annealing (SA), and two hybrid techniques combining GA and SA. Population-based algorithms follow the generic schema of an EA shown below.

ALGORITHM 1 Skeleton of an evolutionary algorithm (EA)

```
1: Initialize(P(0))
2: generation ← 0
3: while not StopCriteria do
4:    Evaluate(P(generation))
5:    Parents ← Selection(P(generation))
6:    Offspring ← Reproduction Operators(Parents)
7:    NewPop ← Replace(Offspring, P(generation))
8:    generation ++
9:    P(generation) ← NewPop
10: end while
11: return Best Solution Ever Found
```

4.1 *Genetic algorithm*

The classical formulation of a GA is given by Goldberg (1989). Based on the generic schema shown in algorithm 1, the GA defines selection, recombination, and mutation operators, and applies them to the population of potential solutions in each generation. In the classical application of a GA the 'reproduction operators' include recombination and mutation.

GA techniques are widely used because of the versatility in solving combinatorial optimization problems. The GA proposed in this work is based on the parallel GA presented by Arraga *et al.* (2003). Implementation details are described in section 6.

4.2 *Simulated annealing*

Simulated annealing is a local search optimization method based on Metropolis's Monte Carlo simulation (Metropolis *et al.* 1953) to find the lowest energy (most stable) orientation for an n-body system. By analogy, the generalization of the Monte Carlo approach to combinatorial problems is straightforward (Kirkpatrick *et al.* 1983). SA maintains a current solution for the problem (analogous to the current state of a system) with an associated objective function

(analogous to the energy function) whose global minimum (analogous to the ground state) is searched.

SA employs a temperature T to control the probability of accepting poorer solutions than the current one. There is no obvious analogy for the temperature T (as such a free parameter does not exist in the combinatorial optimization problem), and so defining an appropriate 'annealing schedule' for avoiding local minima is an art. The parameters for this method, which are often determined emprically, are initial temperature, the number of iterations performed at each step (Markov chain length), and the temperature decreasing schema.

SA is not a population-oriented technique; it maintains only one sub-optimal solution for the problem and explores the search space via certain local search transition operators. Using such operators, it is possible to explore multiple points in the vicinity of the current solution when solving a particular problem. A schema of the SA algorithm based on Ycart (2002) is shown below.

ALGORITHM 2 Simulated annealing algorithm (SA)

```
1: Initialize(T)
2: step ← 0
3: value ← Evaluate(sol)
4: repeat
5:    repeat
6:       step++
7:       newSol ← Generate(sol, T)  // Movement
8:       newvalue ← Evaluate(newsol)
9:       if Accept(value, newvalue, T) then
10:         sol ← newsol
11:         value ← newvalue
12:      end if
13:   until step mod MarkovChainLength) == 0)
14: until StopCriteria
15: return sol
```

4.3 *CHC algorithm*

CHC stands for 'Cross generational elitist selection, Heterogeneous recombination, and Cataclysmic mutation' (Eshelman 1991). CHC is a specialization of a traditional GA which incorporates a very conservative selection strategy, perpetuating the k best individuals, which are always selected to be part of the next generation. In addition, no mutation is applied, and a special cross-over operator is introduced: parents are randomly selected, but only those parents which differ from each other by some number of bits are allowed to reproduce. CHC introduces additional diversity by a re-initialization procedure using the best individual found so far as a template for creating a new population after convergence is detected (*i.e.* when no offspring can be inserted after a number of generations). The initial threshold for allowing mating is often set to a quarter of the chromosome length. If no offspring is inserted into the new population during the mating procedure, this threshold is reduced by 1. The recombination operator in CHC, called uniform cross-over (HUX) performs a special uniform cross-over which randomly swaps exactly half of the bits that differ between the two parent strings.

The algorithm below is a pseudo-code for the CHC algorithm based on Eshelman's proposal. CHC incorporates some features which make it different from traditional GAs:

- the highly elitist replacement strategy
- the use of its own uniform cross-over operator

- absence of mutation, which is substituted by a re-initialization operator
- the use of a mating restriction policy which does not allow recombination of a pair of individuals which are 'too similar'.

ALGORITHM 3 CHC algorithm

```
1: Initialize(P(0))
2: generation ← 0
3: Evaluate(P(0))
4: distance ← chromosomeLength/4
5: while not StopCriteria do
6:    Parents ← Selection(P(generation))
7:    Offspring ← HUX(Parents)
8:    Evaluate(Offspring)
9:    NewPop ← Replace(Offspring, P(generation))
10:   if NewPop == P(generation) then
11:      distance - -
12:   end if
13:   generation ++
14:   P(generation) ← NewPop
15:   if distance == 0 then
16:      Population re-initialization(P(generation))
17:      distance ← chromosomeLength/4
18:   end if
19: end while
20: return Best Solution Ever Found
```

Two variants of the CHC algorithm were developed, differing in the re-initialization mechanism. The initial algorithm (CHC1), uses the SA movement formerly presented as the divergence operator. The analysis of an initial set of results using CHC1 suggested that the re-initialization mechanism would not be able to provide the necessary diversity, given that a large number of modified individuals were discarded as non-feasible solutions. Therefore, an improved CHC version was designed (CHC2). This version forces the feasibility of the solutions generated by the divergence operator, iterating until a feasible individual is obtained.

4.4 *Hybrid algorithms*

In its broadest sense, hybridization refers to the inclusion of problem-dependent knowledge in a general search algorithm (Davis 1991).

One possibility is to construct *strong hybrids* algorithms, where problem knowledge is included as a problem-dependent representation and/or special operators. The other possibility is to combine two or more methods to solve the same problem, constructing *weak hybrids* and trying to take advantage of their salient features to improve the efficiency or accuracy of the new algorithm. The hybrid algorithm defines a new search pattern which determines when each algorithm is executed, and how the internal states of each algorithm report the results so that the other algorithm can continue. Usually, by exchanging a small set of partial solutions or some statistical values, it is possible to combine algorithms in a (hopefully) efficient manner.

In this work, two *weak hybrids* algorithms were designed. In the first type of hybrid (GASA1), one algorithm (GA) uses the other (SA) as an evolutionary operator (figure 4). The rationale for this selection of algorithms is that, while the GA locates 'good' regions of the search space (exploration), the SA allows exploitation in the neighbourhood of these regions. The second hybrid schema executes a GA until the algorithm is completely finished. Then the

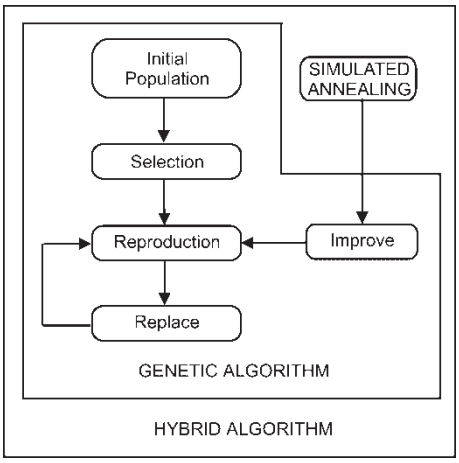

Figure 4. Hybrid schema 1 (GASA1).

hybrid selects some individuals from the final population and executes an SA algorithm over them. A version using tournament selection (GASA2) was analysed (figure 5), although other models using other policies for the selection of individuals are allowed.

4.5 *Parallel algorithms*

Parallel implementations have become popular in the last decade in an attempt to make population-based EAs more efficient. By splitting the population into several processing elements, parallel evolutionary algorithms (PEAs) allow high-quality results to be achieved in a reasonable execution time even for difficult optimization problems (Alba and Tomassini 2002). The parallel implementations for the population-based algorithms (GA, CHC, and hybrids), presented in this work could be categorized within the 'subpopulation with migration' model for EA, according to the classification proposed by Nowostawski and Poli (1999). The original

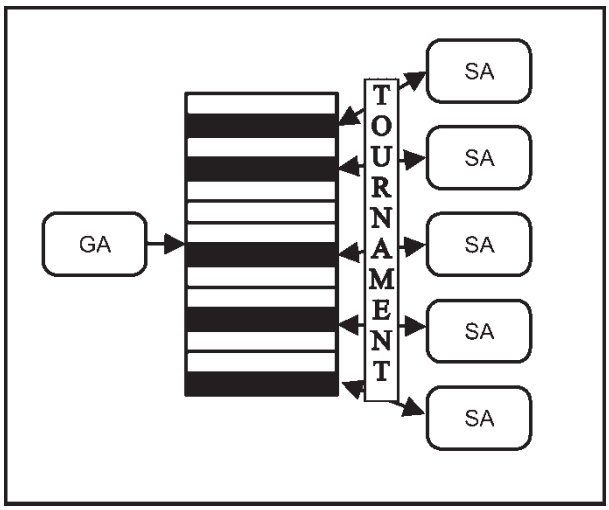

Figure 5. Hybrid schema 2 (GASA2).

population is divided into several subpopulations (*demes*) separated geographically from each other. Each deme runs a serial EA, so that individuals are only able to interact with other individuals in the deme. An additional operator called *migration* is defined: occasionally some selected individuals are exchanged among demes, introducing a new source of diversity in the EA.

The algorithm below shows the generic schema for a population-based PEA.

ALGORITHM 4 Skeleton of a parallel evolutionary algorithm

```
1: Initialize(P(0))
2: generation ← 0
3: while not StopCriteria do
4:    Evaluate(P(generation))
5:    Parents ← Selection(P(generation))
6:    Offspring ← Reproduction Operators(Parents)
7:    NewPop ← Replace(Offspring, P(generation))
8:    generation ++
9:    P(generation) ← NewPop
10:   if SendMigrants then
11:      Migrants ← Selectionformigration(P(generation))
12:      SendMigration(Migrants)
13:   end if
14:   if ReceiveMigrants then
15:      Inmigrants ← ReceiveMigration()
16:      P(generation) ← Insert(Immigrants, P(generation))
17:   end if
18: end while
19: return Best Solution Ever Found
```

Two conditions control the migration procedure: *SendMigrants* determines when the exchange of individuals takes place, and *ReceiveMigrants* establishes whether a foreign set of individuals has to be received or not. These two conditions are separated in time in an *asynchronous* PEA, but they coincide in a *synchronous* model, when the send and receive operations are executed synchronously, one just after the other. *Migrants* denotes the set of individuals to exchange with some other deme, selected according to a given policy. The PEA skeleton explicitly distinguishes between *Selection for reproduction* and *Selection for migration*; they both return a selected group of individuals to perform the operation, but following potentially different policies. The *SendMigration* and *ReceiveMigration* operators carry out the exchange of individuals among demes according to a connectivity graph defined over them, usually a unidirectional ring.

Several approaches to implementing a parallel version for SA have been proposed (Kliewer 2000). The parallel SA presented in this work runs several serial SAs using different initial solutions. The serial algorithms cooperate by sporadically exchanging the best solution found.

5. The MALLBA project

The MALLBA project (Alba *et al.* 2002) is an effort to develop a library of algorithms for optimization which can deal with parallelism (on a local area network (LAN) or a wide area network (WAN)) in a user-friendly and, at the same time, efficient manner. All the algorithms described in the next section are implemented as *software skeletons* in the library. Skeletons are generic templates which are instantiated with the features of the problem by the user.

They incorporate all the knowledge related to the resolution method, its interactions with the problem, and the parallel considerations. Skeletons are implemented by a set of *required* and *provided* C++ classes which represent an abstraction of the entities participating in the resolution method.

- **Provided classes** implement internal aspects of the skeleton in a problem-independent way. The most important *provided* classes are `Solver` (the algorithm) and `SetUpParams` (setup parameters).
- **Required classes** specify information related to the problem. Each skeleton includes the `Problem` and `Solution` required classes, which encapsulate the problem-dependent entities needed by the resolution method. Depending on the skeleton, other classes may be required.

The infrastructure used in the MALLBA project comprises communication networks and clusters of computers located in Málaga, La Laguna and Barcelona, Spain. These sites are connected by a chain of Fast Ethernet and ATM circuits. The MALLBA library is available publicly at the University of Málaga location (`http://neo. lcc.uma.es/mallba/easy-mallba`). By using this library, it was possible to perform a quick coding of different algorithmic prototypes to cope with the inherent difficulties of GSP.

6. GSP encoding

This section explains how to encode a GSP solution into a string of symbols amenable to the optimization algorithms proposed. It also presents the parameter settings used in the experiments, and discusses the fitness function optimized. Finally, some details of the execution platform used are given.

6.1 *Problem encoding*

An edge-oriented binary representation for encoding graphs representing feasible solutions for the GSP was used. A feasible solution is represented as a bit array (indexed from 0 to $|E| - 1$); each bit on the representation indicates the presence or absence of a specific edge existing on the original graph. Figure 6 shows an example graph and its corresponding codification using the proposed edge-oriented binary representation; edges present in the current solution

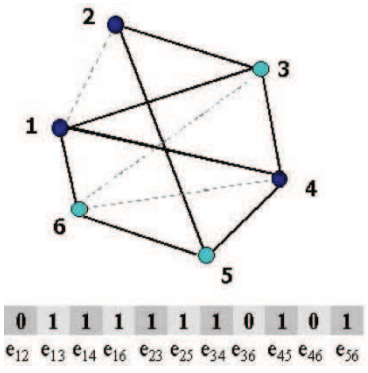

Figure 6. Edge-oriented binary representation.

are drawn using solid lines, while original edges not present in the current solution are drawn using dotted lines.

By using such a binary codification, the evolutionary operators (recombination, mutation) are easy to implement. However, a difficulty must be solved: the operators are able to work out non-feasible solutions. The algorithms presented in this work follow the proposal by Esbensen (1994), discarding non-feasible individuals. This decision simplifies the algorithm, avoiding both quantifying how far those individuals are from the set of feasible solutions and introducing a penalty function for measuring their fitness value.

The feasibility check has two components. A simple heuristic discards a solution if the degree of any terminal node is smaller than its maximum connection requirement. When the degrees of all the terminal nodes are compatible with the connection requirements, the Ford–Fulkerson (1962) algorithm is used to find paths between pairs of terminal nodes, considering one of them as the source and the other as the sink. Since the Ford–Fulkerson algorithm works over directed graphs, each edge is considered as a pair of oriented edges with opposite directions. If a unitary capacity is assumed for each arc, the maximum flow between source and sink matches the maximum number of disjoint paths between the nodes. If the maximum number of disjoint paths is smaller than the corresponding connection requirement, the solution is not feasible.

The population is randomly initialized using a procedure which arbitrarily eliminates up to 5% of the edges from the original graph representation. Then the feasibility check is applied to discard non-feasible initial solutions. Each non-feasible solution detected is dropped from the initial population and the initialization procedure is applied again to generate another one.

6.2 *Parameter settings*

The operators used in the algorithms are as follows.

- GA : proportional selection, two-point recombination, bit-flip mutation.
- CHC: HUX cross-over, elitist selection, re-initialization.
- SA : movement inverts five edges, proportional decaying schema for temperature ($T_k = \alpha \cdot T_{k-1}, \alpha < 1$).

No special configuration analysis has been used to determine the optimum parameter values for each algorithm. Instead, the algorithms worked with the parameter settings derived from the previous work used as a reference baseline (Arraga *et al.* 2003), where population size, mutation, and cross-over probabilities were determined for a GA using the same problem codification.

The population size for all population-based algorithms is 120 individuals. The probability of recombination is 0.9 for GA and for the two GASA variants. The mutation probability has a value of 0.01 in all the algorithms studied.

All the population-based algorithms use the same stop criterion, stopping after reaching a specified number of generations. A high limit value for the number of generations was defined (2000) to provide the algorithms with the chance to reach high-quality results. Use of a convergence detection method was discarded, because such a stop criterion would not lead to a fair comparison of either the results or the execution times.

The SA algorithm uses a proportional decay schema for temperature, with a decay factor value of $\alpha = 0.99$. Since there are not references for SA parameter settings for GSP, after some initial tests working with smaller quantities, values of 250 steps for the Markov chain length and 10 000 iterations were chosen in an attempt to obtain high-quality results.

In the CHC algorithm, the application rate for HUX was fixed at 0.8. The re-initialization procedure involved 35% of the population. The divergence operator used is a variant of the proposed SA movement which modifies up to 40% of alleles with probability 0.5. Use of a simpler divergence operator at low probability (like the GA mutation) would not produce the necessary diversity in the population of CHC after re-initialization.

The two GASA algorithms apply the SA operator with probability 0.01. For GASA1, a short Markov chain is used (10 steps for 20 iterations) in an effort to reduce the computational overhead required to perform the SA as an internal GA operator. For GASA2, where SA is applied after the GA stops, the same Markov chain length is used (10 steps), but the number of iterations is increased to 100.

Population-based PEAs split the population into eight demes and execute them distributed on eight computers. The migration operator sends five individuals to the nearest neighbour, considering the demes to be connected in a unidirectional ring topology. The migration rate value is 25 generations. Policies for both selecting and replacing migrants apply tournaments to a sample of five individuals.

6.3 *Fitness function*

The fitness function evaluates the total cost of the network represented by the solution graph. It is formulated as follows:

$$f = C_{ORIG} - \sum_{i=0}^{|E|-1} EDGE(i) * C(i). \tag{8}$$

The evaluated cost is mapped to a maximization function by subtracting the cost from a constant C_{ORIG}, representing the maximum cost value for the graph considered (when the whole set of edges is included in the solution). In equation (8), $|E|$ is the cardinality of the edge set, the function $C : N \rightarrow R$ returns the cost of an edge, and the function $EDGE : N \rightarrow \{0, 1\}$ returns the binary value corresponding to the edge on the ith position in the representation.

6.4 *Execution platform*

All the algorithms were codified using the MALLBA library implemented on C++. Executions were performed in a cluster of eight Intel Pentium IV machines at 2.4 GHz, each with a 512 Mb RAM, using the SuSE Linux 8.1 operating system connected with a Fast Ethernet LAN at 100 Mbps.

7. Empirical analysis

This section introduces a set of examples designed to evaluate the algorithms implemented. Then, it presents and comments on the experimental results for sequential and parallel versions of the algorithms. Finally, a brief performance analysis is given in which the execution times and the fitness value evolution over generations are examined for the algorithms studied.

7.1 *GSP test suite*

As noted earlier, there is little literature on applying heuristics for solving the GSP and therefore there are no standardized problem benchmarks or test suites. No data are available for real

networks, and since extending existing test suites for related problems (such as the Steiner tree problem) is not easy, the algorithms were evaluated using a random test suite of three problems. Test graphs are identified by reference to the numbers of nodes and terminal nodes (*e.g. graph 50-15* is the smallest graph, with 50 nodes and 15 terminal nodes).

The examples were constructed by randomly selecting the connection requirements on a randomly generated underlying graph topology. Edge cost values are proportional to the Euclidean distance between nodes, except for *graph 100-10* (taken from Arraga *et al.* (2003)) where the edge costs were randomly selected between 0 and 20. The path requirements for each pair of terminal nodes were randomly selected uniformly between 0 and 4 for all three graphs. These test graphs can be considered as 'representatives' for medium size networks with variable terminal connectivity requirements.

Table 2 shows details of the three GSP examples indicating the number of nodes, terminal nodes, and edges, and the connectivity degree (ratio of selected number of edges to number of edges in a complete graph). Table 2 also includes the total cost C_{ORIG} and the previous best known value for each example, (Arraga *et al.* 2003). These values are used as a reference guideline for comparing the accuracy of the EAs considered in this work.

The GSP test suite described and the random graph generator are available at `http://www.fing.edu.uy/inco/grupos/cecal/hpc/gsp`.

Thirty independent runs (with different random seeds) were performed over each of the three graphs in the test suite for serial and parallel algorithms. The results are shown and discussed below.

7.2 *Sequential algorithms*

Table 3 presents the results of the sequential algorithms, showing the best fitness values obtained for each algorithm, as well as the average value and the standard deviation over all 30 independent runs on the three examples studied.

Table 2. GSP test suite.

	graph 100-10	*graph 75-25*	*graph 50-15*
Nodes	100	75	50
Terminals	10	25	15
Edges	500	360	249
Connection degree	0.10	0.13	0.20
C_{ORIG}	4925.0	6294.9	10949.9
Best known value	4561.0	5406.7	9383.0

Table 3. Results of sequential algorithms for the GSP.

graph		GASA1	GASA2	CHC1	CHC2	GA	SA
graph 100-10	Avg	4491.5	4471.5	4449.0	4602.0	4469.5	4186.0
	Best	4549.0	4507.0	4567.0	4634.0	4531.0	4302.0
	Std	36.71	30.22	42.60	15.05	28.17	48.62
graph 75-25	Avg	5363.9	5335.4	3657.0	5478.9	5308.2	5171.8
	Best	5429.3	5427.1	4574.9	5521.8	5406.7	5248.98
	Std	39.8	49.2	372.6	21.2	50.2	51.9
graph 50-15	Avg	9373.3	9330.6	7646.3	9514.2	9316.8	9177.2
	Best	9474.3	9538.3	9143.0	9596.5	9456.0	9321.88
	Std	70.7	85.3	726.2	43.2	85.8	114.6

The analysis of table 3 reveals that the traditional SA algorithm achieves the worst results, being non-competitive compared with the other algorithms. Since the SA algorithm obtains only one solution each time, unlike the population-based algorithms it is not able to take advantage of the diversity to achieve accurate results. SA seems unable to manage the intrinsic complexity of GSP when using the simple transition operator proposed. Additional experiments were performed, using a different neighbourhood definition by changing a greater number of edges (to produce a larger diversity), but the results did not improve. Even when the stop criterion was altered to allow for a larger number of iterations (some experiments using up to 30 000 iterations were performed), the SA algorithm was unable to acheive competitive results.

The canonical GA acheived acceptable results for all the problems studied. However, it did not improve on the best value achieved in *graph 100-10* problem (4561). Also, the average fitness value for the GA is quite far from that former best value, suggesting there is still work to be done in order to fine tune the algorithm.

The performance of the CHC algorithm was unexpectedly efficient and accurate. With a simple configuration and parameter setting, it produced high-quality results. The first variant (CHC1) showed accurate fitness values for *graph 100-10* in the range of GA and GASA results. Nevertheless, it did not behave consistently, showing a lower fitness average and a higher standard deviation than the other algorithms. A possible explanation, as indicated in section 4, is that the restarting procedure applied is too smooth, leading to situations with poor population diversity. Such behaviour was detected in many CHC1 executions over *graph 75-25* and *graph 50-15*, where the algorithm was unable to achieve accurate results and became stuck for many generations because of the similarity between individuals. Thus, the re-initialization mechanism for CHC was enhanced by designing the improved CHC2 algorithm. This algorithm showed better results, significantly exceeding the GA and GASA average and best results for all three graphs. These results suggest that using HUX cross-over is a promising idea deserving further study in the future. CHC2 produced the best value obtained by all algorithms in these experiments for all three graphs, even improving the best results formerly known.

The hybridization technique applied in the GASA1 algorithm worked efficiently, improving the standard GA results, but the one applied in GASA2 usually did not improve them significantly. This suggests that the gradual slight refinement of SA in GASA1 is a more appropriate hybridization policy than using a strong final refinement as in GASA2. In fact, this behaviour was verified when analysing the evolution of best fitness values for some GASA2 executions. A different scenario exists when the SA movement is applied as an inner operator in GASA1, introducing a new source of diversity which helps the GA evolutionary search. Both average and best fitness values are improved in GASA1 hybrid schema compared with the GA

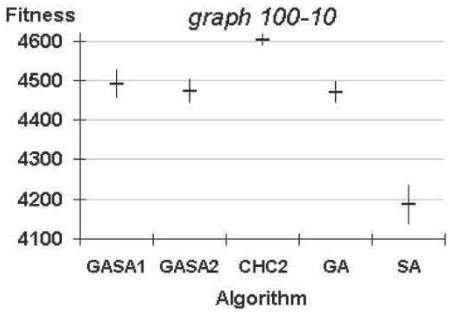

Figure 7. Comparative average results for the *graph 100-10* problem.

results for all three graphs. However, the improvement factor is always less than the standard deviation, thus it cannot be considered a significant enhancement.

Figures 7, 8 and 9 show a graphical comparison of the average results and their standard deviations obtained by the serial algorithms implemented for the three examples considered. CHC1 results have not been included because this algorithm showed premature convergence far from the optimum for *graph 75-25* and *graph 50-15*, as noted earlier. These figures clearly show the superiority of CHC2, the poor accuracy of SA, and the slight improvement showed by GASA1 compared with GA.

7.3 *Parallel algorithms*

Table 4 summarizes the results of the parallel algorithms, showing the best and average fitness values obtained for each algorithm and the standard deviation over all 30 independent runs. The experiments were carried out on the cluster of eight processors described in section 6.4.

Because of the large number of combinations of algorithms and problems, and the run time needed to perform 30 independent executions, only parallel versions of SA and the population-based EAs showing promising behaviour in the serial experiments were executed. The main aim of the study was not to investigate performance issues, but to determine whether the parallel models for EAs could find better solutions than their serial counterparts. It has been observed that demes can explore diverse sections of the search space, introducing a different model of evolution (Cantú-Paz 2000).

The execution times for each algorithm were measured to calculate the speed-up for parallel algorithms implemented over MALLBA. The speed-up values were compared with the almost

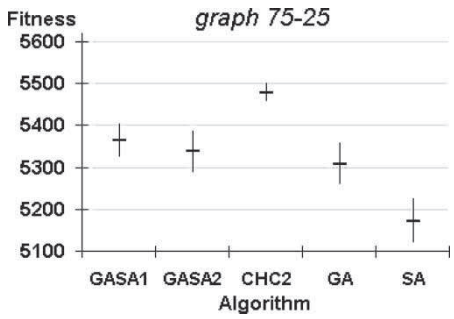

Figure 8. Comparative average results for the *graph 75-25* problem.

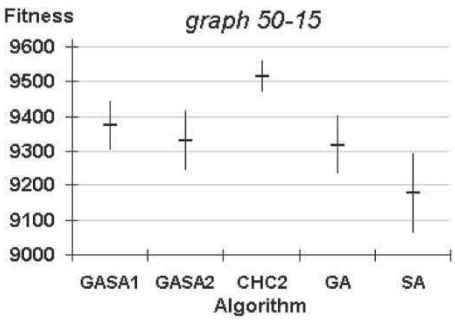

Figure 9. Comparative average results for *graph 50-15* problem.

Table 4. Results of parallel algorithms for GSP.

graph		GASA1	CHC2	GA	SA
graph 100-10	Avg	4529.0	4436.5	4489.0	4170.5
	Best	4608.0	4566.0	4537.0	4289.0
	Std	32.5	51.3	24.4	48.2
graph 75-25	Avg	5415.5	5353.1	5379.4	5174.2
	Best	5440.8	5408.2	5447.2	5290.9
	Std	28.4	34.5	38.8	41.4
graph 50-15	Avg	9473.1	9418.3	9393.2	9266.3
	Best	9569.7	9596.5	9535.3	9440.9
	Std	34.0	70.7	86.7	81.6

linear speed-up behaviour obtained for the former specific GA applied to the GSP (Arraga *et al.* 2003).

Comparison of table 4 with table 3 shows that parallel algorithms improved the best and average fitness results with respect to their serial versions, except for the CHC algorithm. When the population is split into smaller chunks, CHC appears to be unable to improve the high-quality results obtained in serial executions. Rather, parallel CHC produces poor results than serial execution. This effect is consistent with an aspect of PEA behaviour frequently reported in the literature: if the populations are too small, multiple divisions into further smaller groups can be harmful. When working over small populations, the diversity of demes is reduced, and in the cases studied the distributed CHC shows the same difficulties as found on the original CHC algorithm (CHC1). Designing an appropriate parallel CHC algorithm is a clear issue for future work.

Tables 5, 6 and 7 summarize the improvement achieved when using parallel algorithms for each of the instances in the test suite, comparing average and best fitness results obtained in the 30 experiments performed. The improvement factor does not have a high value because the results of the sequential algorithms are already very accurate (because of the large number of generations used as the stopping criterion). However, the improvement on the best and average fitness results is greater than the standard deviation for all graphs studied, showing that the distributed model for evolution produces more accurate results when solving GSP. For GASA1 and GA results where improvements were detected, the Kruskal–Wallis test was performed to analyse the time distributions (*p-values* are shown in the fourth row of the tables).

7.4 *Performance analysis*

This work was not particularly concerned with improvement of the performance as a main objective. However, the execution times for serial and parallel algorithms were also evaluated,

Table 5. Serial versus parallel results, for *graph 100-10 problem.*

	GASA1	CHC2	GA	SA
Serial avg	4491.5	4602.0	4469.5	4186.0
Parallel avg	4529.0	4436.5	4489.0	4170.5
Improvement factor avg	1.008	0.964	1.004	0.996
p-value	1.85×10^{-5}	NA	0.035	NA
Serial best	4549.0	4634.0	4531.0	4302.0
Parallel best	4608.0	4566.0	4537.0	4289.0
Improvement factor best	1.013	0.985	1.001	0.997

Note: NA, not applicable.

Table 6. Serial versus parallel results, for *graph 75-25 problem.*

	GASA1	CHC2	GA	SA
Serial avg	5363.9	5478.9	5308.2	5171.8
Parallel avg	5415.5	5353.1	5379.4	5174.2
Improvement factor avg	1.010	0.977	1.014	1.001
p-value	5.86×10^{-6}	NA	1.93×10^{-6}	0.459
Serial best	5429.3	5521.8	5406.7	5249.0
Parallel best	5440.8	5408.2	5447.2	5290.9
Improvement factor best	1.002	0.980	1.008	1.008

Note: NA, not applicable.

Table 7. Serial versus parallel results, for *graph 50-15 problem.*

	GASA1	CHC2	GA	SA
Serial avg	9373.3	9514.2	9316.8	9177.2
Parallel avg	9473.1	9418.3	9393.2	9266.3
Improvement factor avg	1.011	0.990	1.008	1.010
p-value	8.19×10^{-9}	NA	5.12×10^{-4}	6.56×10^{-5}
Serial best	9474.3	9596.5	9456.0	9321.9
Parallel best	9569.7	9596.5	9535.3	9440.9
Improvement factor best	1.010	1.000	1.008	1.013

Note: NA, not applicable.

Table 8. Comparison of execution times for sequential algorithms.

	$graph\ 100-10$	$graph\ 75-25$	$graph\ 50-15$
GASA1/GA	4.23	3.96	3.69
GASA2/GA	1.13	1.03	1.00
CHC2/GA	0.48	0.89	0.41

and a brief performance analysis was performed. Tables 9, 10, and 11 show the average execution times for sequential and parallel algorithms, measured in minutes.

The execution times show that SA is the fastest algorithm, since it works with only one solution at a time, while population-based algorithms operate on a whole set of individuals. However, SA never reaches accurate solutions. CHC1 shows the same behaviour, except for

Table 9. Average execution times for *graph 100-10* problem.

	GASA1	GA	CHC2	SA	GASA2	CHC1
Serial	520.0	122.8	59.4	18.6	139.2	52.6
Parallel	81.8	24.2	28.7	24	NA	NA
Efficiency	0.79	0.63	0.26	NA	NA	NA

Note: NA, not applicable.

Table 10. Average execution times, for *graph 75-25* problem.

	GASA1	GA	CHC2	SA	GASA2	CHC1
Serial	1229.4	310.2	275.4	31.8	319.8	31.2
Parallel	219.6	59.4	74.4	28.8	NA	NA
Efficiency	0.70	0.65	0.46	0.14	NA	NA

Note: NA, not applicable.

Table 11. Average execution times, for *graph 50-15* problem.

	GASA1	GA	CHC2	SA	GASA2	CHC1
Serial	462.6	125.4	51.0	9.0	126	11.4
Parallel	81.0	24.0	20.4	14.4	NA	NA
Efficiency	0.71	0.65	0.31	NA	NA	NA

Note: NA, not applicable.

the *graph 100-10* problem where it does not become stuck in local optima. For *graph 75-25* and *graph 50-15*, both CHC1 and SA are fast but inaccurate.

The hybrid schema GASA1 is the most complex algorithm, requiring larger computational times. On average, GASA1 runs four times more slowly than the traditional GA. Applying SA after GA termination does not significantly increase the execution time, and so GASA2 has almost the same time demand as GA.

The CHC2 algorithm is even faster than GA. Since CHC2 does not apply continuous mutations and introduces mating restrictions, the effects of superfluous feasibility checks are reduced. This advantage of CHC2 appears to reduce when the number of terminal nodes increases, because the ratio $avgtime$(CHC)/$avgtime$(GA) ranges from 0.48 to 0.89 when the number of terminal nodes increases from 10 to 25. Table 8 shows comparative execution times for population-based sequential algorithms, using the GA execution time as reference.

These results support the conjecture that the complexity of GPS is mainly related to the number of terminal nodes, and is only slightly affected by the total number of nodes. This can be seen in a comparison of the execution times of a given algorithm for the three instances: the largest effort is always required for solving *graph 75-25*. This behaviour can be explained by considering that the feasibility check, which demands most of the computational effort used in every generation, has a complexity proportional to the square of the number of terminal nodes ($O(n_T^2)$).

An intuitive value for the speed-up achieved when running parallel algorithms over eight machines was also computed. Parallel versions required less time than serial versions because they worked with reduced populations. Values of $Efficiency = Speedup/Number\ Of\ Machines$ are included in tables 9, 10, and 11. Although all algorithms showed sublinear speed-up behaviour, both GASA1 and GA have a high value for efficiency, in the range of 0.7. CHC2 showed a poor efficiency when working with distributed populations, nearly 0.3 for graphs with few terminal nodes, and ranging up to 0.46 for *graph 75-25*. This shows once more that parallel CHC does not have the same pattern of behaviour evidenced by other population-based parallel EAs.

7.5 *Tracking the fitness evolution*

This subsection analyses an important aspect of EAs: the evolution of fitness value over the generations. Figures 10, 11, and 12 show the evolution of the best fitness values observed for sequential versions of CHC2, GASA1, GA, and SA during representative samples run over *graph 100-10*, *graph 75-25*, and *graph 50-15* problems respectively.

GASA2 and CHC1 have been intentionally omitted from the analysis, because GASA2 has the same fitness evolution behaviour as GA as there is no difference between their internal operations until GA stops, while CHC1 has the same behaviour as CHC2 until it becomes stuck due to premature convergence. Zoomed graphics showing the fitness evolution in the first 300 generations are presented to appreciate better the differences between values.

Over *graph 100-10*, both the hybrid GASA1 algorithm and CHC2 are able to compute well-suited individuals in a moderately low number of generations. Moreover, they reach relative

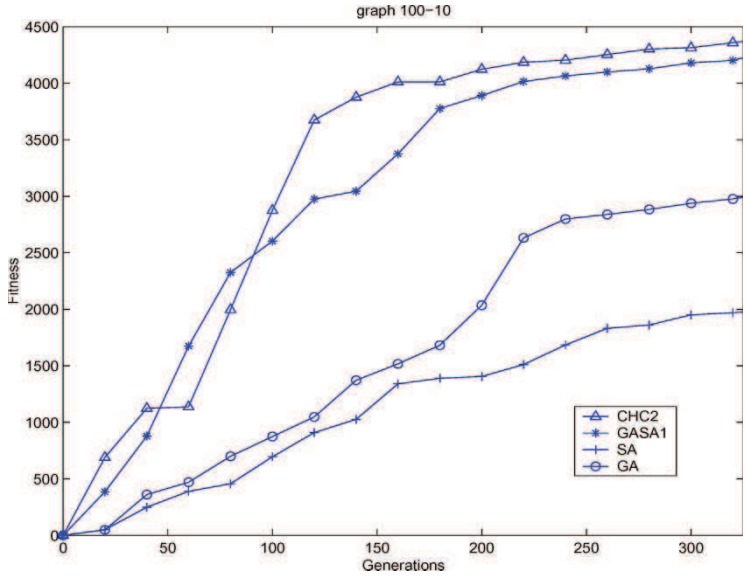

Figure 10. Fitness evolution for the *graph 100-10* GSP.

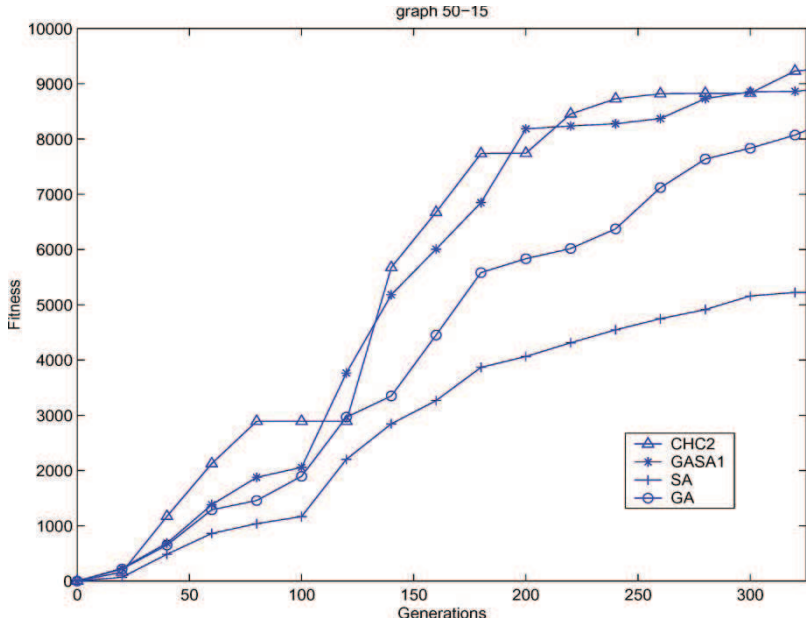

Figure 11. Fitness evolution for the *graph 75-25* GSP.

high fitness values in less than 200 generations. GA works more slowly, usually needing more generations before finding comparable results. SA shows the worst behaviour, following a more lethargic fitness evolution pattern. However, this behaviour is not the same over *graph 75-25* and *graph 50-15*. Although GASA1 and CHC reach high fitness values faster, the gap between their results and GA is not so large. Furthermore, over *graph 50-15* problem the differences are significantly reduced. Once again, SA shows the worst behaviour on every problem.

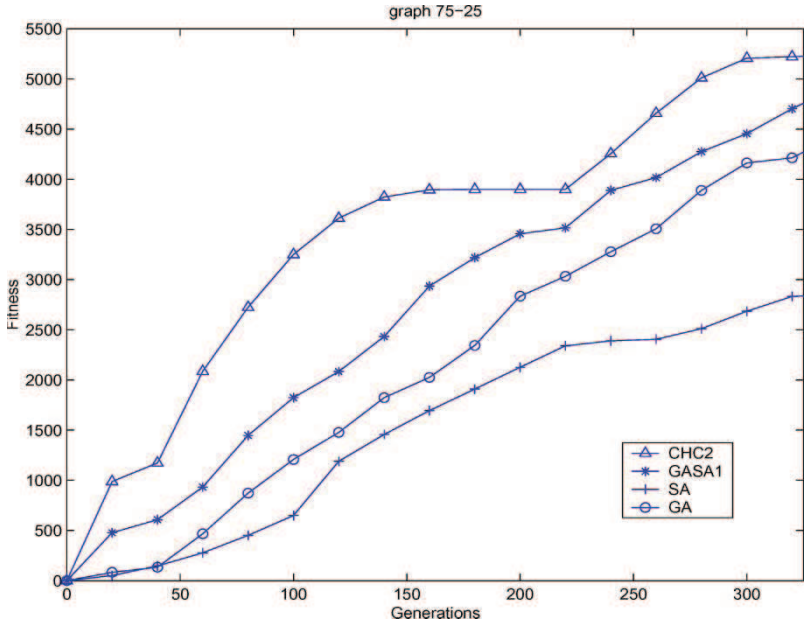

Figure 12. Fitness evolution for the *graph 50-15* GSP.

8. Conclusions

This work evaluates several EAs applied to the GSP problem. Serial and parallel versions of the algorithms were codified and executed using the MALLBA library. It includes a study of the algorithm behaviour and a performance discussion when solving three examples of the problem.

Analysing the results of serial and parallel algorithms for the test suite used, some conclusions can be drawn on the applicability of SA, CHC, GA, and the two hybrid algorithms studied.

The SA algorithm was found not to be well suited to solving the GSP when using a simple movement (neighbourhood inspection) operator. Even when a very large number of iterations were allowed, SA was unable to reach competitive results with respect to the other methods.

Even though from a numerical point of view (average best fitness values), similar values were obtained for GA and the two hybrid algorithms some important details should be noted. GASA2, applying SA on the final population of a GA, yielded poor results, showing a negligible improvement over GA. However, GASA1, using SA as an inner GA operator, found better solutions and accelerated the search. Yet GASA1 improvements are not important in the long run. Its average best fitness value is very close to the GA result, and so the SA operator only contributes with a rapid initial exploration of the search space, at the cost of heavily increasing the execution time. In addition, the pure GA algorithm rarely became stuck in local optima when solving the GSP; thus the SA capability for accepting worse solutions is not a significant feature.

CHC was found to be a very promising algorithm for solving the GSP problem class, possibly taking advantage of its special HUX cross-over. This algorithm gives accurate individuals in a low number of generations and is able to improve its best fitness value faster than the other algorithms studied. The first version (CHC1) often becomes stuck because of the restricted re-initialization operator used. When this operator was improved, a more robust

algorithm (CHC2) was obtained which achieved highly accurate results for all three instances of GSP studied. The algorithm improved the best values found using any of the other algorithms, produced significantly improved best average values, and even showed more efficient computational performance than the GA and the hybrid algorithms.

The study confirmed that the parallel model for EAs is able to find better solutions for the GSP, giving slight improvements over the sequential models. In addition, performance is improved when populations are split. Although all algorithms had sublinear speed-up behaviour, both GASA1 and GA showed high efficiency when using eight demes and are amenable to parallelism.

Two main areas of future work are further analysis of the algorithm behaviour and enhancing the efficiency.

The issue related to algorithm behaviour includes studying the hybridization mechanisms in order to determine their contributions when solving the problem, particularly considering the extra computational effort demanded by the hybrids. Further work has to be done to investigate the influence of the CHC divergence operator, given that when not enough diversity is provided (particularly in the distributed model), the algorithm showed premature convergence. The divergence operator used in CHC2 is not efficient, since it has to be applied several times until a feasible solution is produced. Designing an efficient divergence operator which ensures diversity and feasibility in population individuals is not a trivial task, and both theoretical and experimental contributions will be of importance.

However, there is room for improvement in the efficiency of both serial and parallel algorithms. The implementations used in this work were not optimized, and so they spend some time in superfluous feasibility checks which slow down all algorithms. Improvements in this area are necessary to solve larger and more complex GSP, for which the current algorithms would need a long time. Related to this point, the scalability of the distributed population algorithms proposed could be further investigated to determine whether it will be useful to solve very complex GSP by using the computational power of larger clusters of machines.

Acknowledgements

SN is partly supported by Comisión Sectorial de Investigación Científica, Universidad de la República, Uruguay, under project Parallel Genetic Algorithms and their Application to Reliable Communication Networks Design. HC is partly supported by INRIA (external team PAIR) and by CNPq–Projeto PROSUL Proc. 490333/2004-4. EA is partially funded by Ministerio de Ciencia y Tecnología and FEDER under contract OPLINK, TIN2005-08818-C04-01.

References

Alba, E., and Tomassini, M., Parallelism and evolutionary algorithms. *IEEE Trans. Evolut. Comput.*, 2002, **6**(5), 443–462.

Alba, E., Almeida, F., Blesa, M., *et al.*, Mallba: a library of skeletons for combinatorial optimisation, in *Proceedings of the Euro-Par*, 2002, pp. 927–932.

Arraga, S., Aroztegui, M., and Nesmachnow, S., Resolución del problema de Steiner generalizado utilizando un algoritmo genético paralelo, in *3er Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB '03)*, 2003, pp. 387–394 (in Spanish).

Bäck, T., Fogel, D.B. and Michalewicz, Z. (Eds) *Handbook of Evolutionary Computation*, 1997 (Oxford University Press: Oxford).

Ball, M.O., Computing network reliability. *Oper. Res.*, 1979, **27**, 832–836.

Cantú-Paz, E., *Efficient and Accurate Parallel Genetic Algorithms*, 2000 (Kluwer Academic: Dordrecht).

Corne, D., Oates, M. and Smith, G., *Telecommunications Optimization*, 2000 (John Wiley: New York).

Davis, L., *Handbook of Genetic Algorithms*, 1991 (van Nostrand–Reinhold: New York).

Davis, L., Orvosh, D., Cox, A. and Qui, Y., A genetic algorithm for survivable network design, in *Proceedings of the 5th International Conference on Genetic Algorithms*, pp. 408–415, 1993 (Morgan Kaufman: San Mateo, CA).

Esbensen, H., Computing near-optimal solutions to the Steiner problem in a graph using a genetic algorithm. PhD thesis, University of Aarhus, 1994.

Esbensen, H. and Mazumder, P., A genetic algorithm for the Steiner problem in a graph, in *EDAC-ETC-EUROASIC*, pp. 402–406, 1994.

Eshelman, L., The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination. In *Foundations of Genetics Algorithms*, pp. 265–283, 1991 (Morgan Kaufmann: San Mateo, CA).

Ford, L., and Fulkerson, D., *Flows in Networks*, 1962 (Princeton University Press: Princeton, NJ).

Galiasso, P. Wainwright, R., A hybrid genetic algorithm for the point to multipoint routing problem with single split paths, in *ACM Symposium on Applied Computing (SAC)*, 2001, pp. 327–332.

Goldberg, D. *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1989 (Addison Wesley: Reuding, MA).

Hesser, J., Manner, R. and Stucky, O., Optimization of Steiner trees by genetic algorithms, in *Proceedings of the 3rd International Conference on Genetic Algorithms*, 1989, pp. 231–236 (Morgan Kaufman: San Mateo, CA).

Hiramatsu, A., Shimamoto, N. and Yamasaki, K., A dynamic routing control based on a genetic algorithm, in *Proceedings of International Joint Conference on Neural Networkds (IJCNN)*, 1993, pp. 1123–1128.

Hwang, R., Do, W. and Yang, S., Multicast routing based on genetic algorithms. *J. Inform. Sci. Eng.*, 2000, **16**(6), 885–901.

Julstrom, B., A genetic algorithm for the rectilinear Steiner problem, in *Proceedings of the 5th International Conference on Genetic Algorithms*, pp. 474–480, 1993 (Morgan Kaufman: San Mateo, CA).

Julstrom, B., Seeding the population: improved performance in a genetic algorithm for the rectilinear Steiner problem, in *ACM Symposium on Applied Computing (SAC)*, 1994, pp. 222–226.

Julstrom, B., Encoding rectilinear Steiner trees as lists of edges, in *ACM Symposium on Applied Computing (SAC)*, 2001, pp. 356–360.

Kahn, V. and Crescenzi, P., A compendium of NP optimization problems, Available online at: http;//www.nada.kth.se/theory/problemlist.html (accessed December 2006).

Kapsalis, A., Rayward-Smith, V. and Smith, G., Solving the graphical Steiner tree problem using genetic algorithms. *J. Oper. Res. Soc.*, 1993, **44**, 397–406.

Karp, R., Reducibility among combinatorial problems. In *Complexity of Computer Communications*, pp. 85–103, 1972 (Plenum Press: New York).

Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P., Optimization by simulated annealing. *Science*, 1983, **220**, 671–680.

Kliewer, G., A general software library for parallel simulated accealing, in *EURO Winter Institute on Metaheuristics in Combinatorial Optimisation*, 2000.

Ljubic, I. and Kratica, J., A genetic algorithm for biconnectivity augmentation problem, in *Proceedings of the 2000 IEEE congress on Evolutionary Computation*, pp. 89–96, 2000 (IEEE Press: New York).

Ljubic, I., Raidl, G. and Kratica, J., A hybrid GA for the edge-biconnectivity augmentation problem, in *Proceedings of the 2000 Parallel Problem Solving from Nature VI Conference*, pp. 641–650, 2000 (Springer Velag: Berlin).

Metrȯpolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H. and Teller, E., Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 1953, **21**, 087–1092.

Nowostawski, M. and Poli, R., Parallel genetic algorithms taxonomy, in *Proceedings of the 3rd International Conference on Knowledge-Based Intelligent Information Engineering Systems*, pp. 88–92, 1999 (IEEE: New York).

Pedrycz, W. and Vasilakos, A., *Computational Intelligence in Telecommunications Networks*, 2001 (CRC Press: Boca Raton, FL).

Robledo, F., Diseño topologico de redes, casos de estudio. (The generalized Steiner problem y the Steiner 2-edge-connected subgraph problem)Master thesis, PEDECIBA Informaática, Universidad de la República, Uruguay, 2001 (in Spanish).

Wakabayashi, S., A genetic algorithm for the rectilinear Steiner tree problem in VLSI interconnect layout. *IPSJ J.*, 2002, **43**, 5–19.

Winter, P., Steiner problem in networks: a survey. *Networks*, 1987, **7**(2), 129–167.

Xianwei, Z., Changjia, C. and Gang, Z., A genetic algorithm for multicasting routing problem, in *Proceeding of the International Conference on Communication Technology (ICCT)*, 2000.

Ycart, B., *Modd́les et Algorithmes Markoviens*, 2002 (Springer, New York).

Zhu, L., Schoenefeld, D. and Wainwright, R., A genetic algorithm for the point to multipoint routing problem with varying number of requests, in *Proceedings of the 1998 IEEE International Conference on Evolutionary Computing (ICEC'98)*, 1998.