Journal of Universal Computer Science, vol. 14, no. 15 (2008), 2491-2511
submitted: 5/2/08, accepted: 30/6/08, appeared: 1/8/08 © J.UCS

A Hybrid Transgenetic Algorithm for the Prize Collecting Steiner Tree Problem

Elizabeth Ferreira Gouvêa Goldbarg
(Federal University of Rio Grande do Norte, Natal, Brazil
beth@dimap.ufrn.br)

Marco César Goldbarg
(Federal University of Rio Grande do Norte, Natal, Brazil
gold@dimap.ufrn.br)

Cristine Cunha Schmidt
(Federal University of Rio Grande do Norte, Natal, Brazil
cristines@gmail.com)

Abstract: Evolutionary algorithms are effective search tools for tackling difficult optimization problems. In this paper an algorithm based on living processes where cooperation is the main evolutionary strategy is applied to the Prize Collecting Steiner Tree Problem, an NP-hard combinatorial optimization problem. The Transgenetic Algorithm presented here is hybridized with path-relinking. Computational results of an experiment performed with benchmark instances are reported. The results obtained for the Prize Collecting Steiner Tree Problem with the application of the hybrid Transgenetic Algorithm are compared with the results of three effective approaches presented previously. The computational experiment shows that the proposed approach is very competitive concerning both quality of solution and processing time.

Keywords: Prize Collecting Steiner Tree Problem, Transgenetic Algorithm, Evolutionary Algorithm, Path-relinking
Categories: I.2.8, G.2.3

1 Introduction

Evolution has been a source of inspiration for developing problem solving methods to difficult optimization problems. The underlying idea concerning the design of computational techniques based on natural evolution is to develop stochastic search methods that operate on a population of candidate solutions. These methods are called Evolutionary Algorithms. The search operators used by classical evolutionary algorithms are inspired on neo-Darwinian evolutionary mechanisms whose typical rules are: selection, crossover and mutation [Mitchell 1998]. As a source of inspiration, nature has led to the emergence of a wide variety of types of evolutionary algorithms. In this paper an evolutionary algorithm whose metaphor is based on two driving forces of evolution is applied to the Prize Collecting Steiner Tree Problem, an NP-hard combinatorial optimization problem. The proposed heuristics belongs to the class of Transgenetic Algorithms. The algorithms of this class are inspired on the endosymbiosis theory of evolution and on natural horizontal gene transfer mechanisms [Goldbarg et al. 2007].

The term "endosymbiosis" specifies the relationship between organisms which live one within another (symbiont within host). The endosymbiotic theory of evolution [Margulis 1991] deals with the concept of symbiogenesis that implies the appearance of new tissues, new organs, physiologies or other new features that result from the endosymbiotic association. In this theory cooperation rather than competition is emphasized as a major evolutionary force [Margulis and Sagan 1986]. Horizontal gene transfer between the endosymbionts and the host plays an important role in endosymbiosis. The horizontal gene transfer is defined to be the movement of genetic material between organisms other than by descent in which information travels through the generations as the cell divides. The horizontal gene transfer in an endosymbiotic process is called endosymbiotic gene transfer [Timmis et al. 2004].

The instigating scenario described above led to the development of Transgenetic Algorithms, where the concepts of that evolution theory are adapted to the computational context in order to search the solution space of optimization problems. These algorithms deal with a population of candidate solutions, called endosymbiont chromosomes, a population of entities that modify the endosymbiont chromosomes, called transgenetic vectors, and information obtained before and during the search process that is thought to be in a host cell. Unlike many other evolutionary approaches, the chromosomes of Transgenetic Algorithms do not share genetic material directly by means of recombination operations. Chromosomes are uniquely modified by the transgenetic vectors which are the main intensification and diversification tools of the Transgenetic Algorithms. The information utilized by the transgenetic vectors to alter the chromosomes is the host's information. The information sources include *a priori* and *a posteriori* knowledge about the problem and about the search executed by the algorithm, respectively.

The Transgenetic Algorithm presented here is hybridized with a path-relinking procedure. Path-relinking was originally proposed in the context of Tabu Search [Glover 1994]. Given two solutions, the idea behind this strategy is to generate new solutions exploring trajectories between the two initial solutions. One of these two solutions is chosen to be the origin solution, x_o, and the other is the target solution x_t. The roles of origin and target can be interchangeable. Starting from x_o, the objective is to generate a path in the solution space that leads toward new solutions, also called guiding solutions. The attributes of the target solution are iteratively introduced in the origin solution leading to a sequence x_o, $x_o(1)$, $x_o(2)$, …, $x_o(r) = x_t$, where $x_o(i+1)$ is obtained from $x_o(i)$ by a move that introduces in $x_o(i+1)$ an attribute that reduces the distance between attributes of the origin and target solutions. Examples of these attributes for the Prize Collecting Steiner Tree Problem include nodes and edges of the instance graph. A number of metrics can be utilized to measure the distance between solutions [Sörensen 2007].

This paper is organized as follows. In section 2 the Prize Collecting Steiner Tree Problem is introduced and a review of the methods proposed to solve this problem is presented. Section 3 presents the general architecture of a Transgenetic Algorithm. Section 4 presents the hybrid algorithm proposed to solve the investigated problem. A computational experiment with 114 benchmark instances is reported in section 5. The proposed algorithm is compared with three state-of-the-art algorithms for the Prize Collecting Steiner Tree Problem. The data obtained with the experiments show that the proposed algorithm is very competitive concerning both quality of solution and

processing time. Finally, some conclusions and future works are presented in sections 6 and 7, respectively.

2 The Prize Collecting Steiner Tree Problem

Let $G = (V, E, c, w)$ be a connected and undirected graph where $V = \{v_1, \ldots, v_n\}$ is the vertex set, $E = \{e_1, \ldots, e_m\}$ is the edge set, c: $E \rightarrow \Re^+$ and w: $V \rightarrow \Re^+$ are non negative cost functions associated with the edge and node sets, respectively. The problem consists in finding a subgraph $T = (V_T, E_T)$, $V_T \subseteq V$, $E_T \subseteq E$, that minimizes the objective function described in equation 1 [Goemans and Williamson 1996].

$$f(\mathrm{T}) = \sum_{v \notin V_T} w(v) + \sum_{e \in E_T} c(e) \tag{1}$$

The investigated problem is a generalization of the classical Steiner Tree Problem. The PCSTP models a number of real world applications, mainly on the design of infra-structure networks, such as gas, water, electricity and telecommunications [Canuto et al. 2001, Johnson et al. 2000, Ljubić et al. 2004]. The general scenario is a set of potential customers demanding services that are network distributed. When considering the installation of such a network companies have to deal with the trade-off between the sum of potential profits over the selected customers and network costs. For instance, consider a network to distribute natural gas for customers in an urban area. The graph corresponds to a street map. The edges represent the street segments where pipes will be further laid. The nodes represent the location of potential customers and street intersections. The cost assigned to each node is an estimative of the financial loss that would result if the customer represented by that node is not in the selected set of customers. Value $w(v) = 0$ is assigned to every node that represents a street intersection. The cost $c(e)$ assigned to edge e corresponds to the cost of laying a pipe on the street segment represented by e. Utilizing this model, companies search for networks of customers that are financially attractive.

The first time the Steiner tree problem with weights on nodes was investigated dates back to 1987 [Segev 1987]. It was shown that if the weights are non-negative and the root of a solution tree is known, then the problem corresponds to a directed Steiner tree problem. Another NP-hard problem was also presented in the same paper, the single point weighted Steiner tree problem, where a new point has to be included in a pre-existent solution. The term "prize collecting" [Balas 1989] was introduced in the context of the Traveling Salesman Problem. In that version node weights represent penalties that have to be added to the cost of the traveling salesman tour when the node is not included in the tour. The PCSTP was introduced in a paper where a 3-approximate algorithm was proposed tackle the problem [Bienstock et al. 1993].

Once the PCSTP is largely applicable to a number of real world situations, several algorithms have been presented to solve it. Many of those algorithms are tested in the set of benchmark instances available at http:www.research.att.com/~mgcr/data/index.html. This database contains four classes of instances [Johnson et al. 2000, Canuto et al. 2001]. Class K is composed of geometric graphs randomly generated designed to have a structure similar to a street

map, with 100 to 400 nodes and 319 to 1507 edges. Class P is composed of non-structured graphs with 100 to 400 nodes and 284 to 11144 edges. The instances of classes C and D are derived from Steiner tree benchmark instances. The instances of class C have 500 nodes and 625 to 12500 edges. The instances of class D have 1000 nodes and 1250 to 25000 edges. Other instances also proposed for the PCSTP represent real examples of networks of a German city [Ljubić et al. 2004].

The first exact algorithm proposed for the PCSTP was a cutting-plane algorithm based on an integer programming formulation with a set of constraints called *generalized subtour elimination constraints* [Lucena and Resende 2004]. Those constraints were a generalization of the subtour elimination constraints [Dantzig et al. 1954]. The exact algorithm was applied to the 114 instances of classes K, P, C and D solving 96 of them. An effective branch-and-cut algorithm [Ljubić et al. 2006] solved all instances of classes K, P, C, D and the instances that represent real networks of a German city [Ljubić et al. 2004]. The same set of instances was solved by another branch-and-cut algorithm based on Lagrangian and Linear Programming relaxations [Cunha et al. 2008]. In the same paper, the authors presented a heuristic method based on the dual bound obtained with the Lagrangian relaxation.

Due to the large applicability of the PCSTP, a number of approximation algorithms were proposed for it [Bienstock et al. 1993, Goemans and Williamson 1996, Johnson et al. 2000, Feofllofl et al. 2007]. A multi-start algorithm with Variable Neighborhood Search and Path-relinking [Canuto et al. 2001] was applied to the instances of classes K, P, C and D. The initial solutions of the multi-start algorithm were built with a primal-dual method [Goemans and Williamson 1996] with perturbations. Those solutions were improved in a local search procedure. A Variable Neighborhood Search with Path-relinking was used as a post-optimization step. A Memetic Algorithm hybridized with a relaxation of an Integer Programming model [Klau et al. 2004] was also presented to tackle the investigated problem. This algorithm was also applied to the instances of classes K, P, C and D and the results were compared with the ones obtained with the multi-start algorithm.

Some variants of the PCSTP are named The Net Worth Maximization Problem, the Quota Problem and the Budget Problem [Johnson et al. 2000]. The Net Worth Maximization Problem consists in finding a subtree $T = (V_T, E_T)$, $V_T \subseteq V$, $E_T \subseteq E$, that maximizes the objective function described in equation 2. Given a prize quota $Q > 0$, the Quota Problem consists in finding a subtree $T = (V_T, E_T)$ that minimizes $\sum_{e \in E_T} c(e)$ subjected to $\sum_{v \in V_T} w(v) \geq Q$. Finally, the Budget Problem consists in, given an edge budget $B > 0$, finding a subtree $T = (V_T, E_T)$ that maximizes $\sum_{v \in V_T} w(v)$, subjected to $\sum_{e \in E_T} c(e) \leq B$.

$$NW(T) = \sum_{v \in V_T} w(v) - \sum_{e \in E_T} c(e) \tag{2}$$

In a generalization of the Quota Problem [Haouari et al. 2008], the vertex set V is partitioned in K+1 nonempty disjoint subsets $V_0, V_1, \ldots, V_K$ such that, $V_0 = \{v_0\}$ and for each subset V_k, $k = 1, \ldots, K$, a prize quota $Q_k > 0$ is defined. A non negative penalty function is associated with each vertex $v \in V \backslash V_0$, $\gamma: V \backslash V_0 \rightarrow \Re^+$. The

generalized Quota Problem consists in finding a subtree $T = (V_T,E_T)$ that minimizes $\sum_{v \notin V_T} \gamma(v) + \sum_{e \in E_T} c(e)$, subjected to $v_0 \in V_T$ and $\sum_{v \in \{V_T \cap V_k\}} w(v) \geq Q_k$.

3 Transgenetic Algorithms

Transgenetic Algorithms are evolutionary techniques whose biological inspiration comes from the endosymbiotic theory of evolution. That theory states that a new organism can emerge from the fusion of two or more independent beings [Margulis 1991]. The term "endosymbiosis" specifies the relationship between organisms which live one within another (symbiont within host) in a mutually beneficial relationship. Observing that the DNA is not only in the nucleus of cells, the biologist Margulis proposed that eukaryotic cells originated as communities of interacting entities that joined together. Those entities became, later, the organelles of a single host. Today, researchers recognize the horizontal transfer of functional genes between organisms as a determinant factor of the endosymbiotic origin of cellular organelles [Pierce et al. 2003]. Two natural vehicles of horizontal gene transfer are called plasmids and transposons. Plasmids are mobile genetic particles, that is, DNA rings that can be exchanged between certain cells and that can replicate independently of the chromosome. Transposons or “jumping genes” are genetic elements that can spontaneously move from one position to another in a DNA molecule. Some microbiologists believe that, during bacterial evolution, the ability of bacteria to adapt to new environments most often results from the acquisition of new genes through horizontal transfer rather than by the alteration of gene functions through numerous point mutations.

Transgenetic Algorithms are evolutionary algorithms whose context is thought to occur in a cell, where a population of endosymbionts co-evolves with its host, improving the fitness of the system endosymbionts/host [Goldbarg et al. 2007]. The evolutionary process is accomplished by means of genetic rearrangement and exchanging between the host cell and the endosymbiont chromosomes. Biologists argue that the term *chromosome* does not apply to bacteria, although it is often used [Margulis 2004]. The reason for that concerns significant differences in the DNA structure of bacteria. The appropriate term is *chromoneme*. However, in order to avoid introducing a new nomenclature in the context of evolutionary algorithms when referring the elements of a population, the authors maintain the term *chromosome* in this paper.

During the search process executed by the Transgenetic Algorithms, the genetic codes of the endosymbionts are modified. The alterations made in the chromosomes are accomplished by means of manipulation vectors that mimic the action of natural vehicles of horizontal gene transfer, such as plasmids and transposons.

The population of endosymbiont chromosomes is the base of the search process. Alike another evolutionary algorithms, chromosomes represent problem solutions. The chromosomes are manipulated by the transgenetic vectors, the second component of the search process. Those vectors are the main tools for search intensification and diversification. The transgenetic vectors have “instructions” of how to manipulate the code of chromosomes. Rather than breeding to generate offspring, the random variation of solutions is uniquely operated by means of these vectors. Thus, unlike

other evolutionary approaches, the chromosomes of Transgenetic Algorithms do not share genetic material directly. The third component of the algorithm is the host, where information about the problem being tackled (*a priori* information) and information about the search process (*a posteriori* information) is thought to be stored. This final component of the algorithm is called *host's repository*. The information stored in the host's repository is used in the transgenetic vectors to manipulate the chromosomes. As a result of the manipulation by the transgenetic vector, the codes of chromosomes are modified, yielding the random variation necessary for searching the space of solutions of the investigated problem. To summarize, three contexts are considered in Transgenetic Algorithms:

- The endosymbiont chromosomes: a population of candidate solutions
- The host's repository: a base of information about the problem or about the search.
- The transgenetic vectors: entities that modify the candidate solutions, transporting information from the host's repository to the population of chromosomes or, simply, rearranging the genetic code of chromosomes.

The three contexts together are thought to form a small ecosystem where the evolutionary process occurs. Rather than be subjected to an external environmental pressure, the evolution in this small ecosystem is guided and directed to the absorption of the endosymbionts.

1. Generate and evaluate an initial population of chromosomes
2. Initialize the host's repository (HR)
3. Repeat
4. Generate transgenetic vectors
5. Select chromosomes for manipulation
6. Manipulate chromosomes
7. Update HD
8. until a stopping criterion is satisfied

Figure 1: Framework of a Transgenetic Algorithm

Figure 1 shows the general steps of a Transgenetic Algorithm. Initially, a population of candidate solutions is generated and evaluated. The host's repository with information to be used during the search is initialized in step 3 and updated in step 7. This repository is initialized with *a priori* information about the investigated problem. This information regards known lower or upper bounds of the problem, heuristic solutions, information obtained from statistical analysis of the problem instance, among others. *A posteriori* information is obtained during the algorithm execution, such as solutions or partial solutions obtained during the search and information obtained from statistics of the population.

In step 4 the transgenetic vectors are generated. A transgenetic vector, λ, is a pair $\lambda = (I, \Phi)$, where I stands for information and Φ is a method to manipulate the genetic code of chromosomes. The method Φ is formed by a set of procedures, $\Phi = (p_1,...,p_s)$. The procedures that form the method of a given transgenetic vector depends on the

type of the vector. In this paper, two transgenetic vectors are utilized: plasmids and transposons.

The information used in the plasmids is structured as in the chromosomes. Thus, the information of plasmids represents partial solutions. The size of a given plasmid is the length of its information string. The manipulation method of the plasmids contains two procedures: p_1 and p_2. Procedure p_1, called *attack*, verifies whether a chromosome C is susceptible or not to the manipulation of a transgenetic vector λ. Thus, procedure p_1 implements a function A: C, $\lambda \rightarrow$ {false,true}. If p_1 returns "true", then the information string of the plasmid λ is transcribed into the chromosome C. Procedure p_2 defines how the transcription is done, that is, the steps that describe how to transfer the information of the plasmid to the chromosome are defined in p_2.

Transposons are transgenetic vectors whose information is a rule for gene rearrangement. Besides procedures p_1 and p_2 defined previously, the manipulation method of transposons utilizes a third procedure, p_3 that identifies the positions in the target chromosome that will be manipulated by the transgenetic vector, that is, the genes that will be rearranged.

A subset of chromosomes is selected to be manipulated by one or more transgenetic vectors in step 5 of the algorithm shown in figure 1. The manipulation of the chromosomes by the transgenetic vectors (step 6) may generate new interesting information for the evolutionary process, for example, new best solutions. In this case the host's repository is updated with the new information (step 7). The algorithm iterates until a stopping condition is reached (step 8).

Concerning the update of the host's repository, it is important to make clear that full solutions, such as the ones represented in chromosomes, are never utilized in their totality to compose the information of the transgenetic vectors. Existing solutions that are selected to update the host's repository are used as information sources, only fragments of those solutions are utilized by the transgenetic vectors.

Mechanisms of horizontal gene transfer and endosymbiotic interactions, separately, were sources of inspiration for other evolutionary algorithms. There are two general directions followed by previous approaches that deal with the concept of horizontal gene transfer mechanisms. The algorithms that follow the first direction are standard Genetic Algorithms increased with operations based on some lateral gene transfer mechanism [Chan et al. 2005, Yeung et al. 2008]. The other direction followed for developing evolutionary algorithms concerns the substitution of crossover operations in standard Genetic Algorithms by operations based on horizontal gene transfer mechanisms [Perales-Graván and Lahoz-Beltra 2008]. The algorithms based on endosymbiotic interactions are variations of cooperative co-evolutionary algorithms [Kim et al. 2001, 2006]. They have different populations that consist of partial solutions of the investigated problem and another population that consists of complete solutions. The populations of partial solutions combine to form the one that consists of complete solutions. Each population evolves, separately, by means of a standard genetic algorithm. Then the individuals of each population of partial solution compete with another of the population of complete solutions in order to obtain new individuals of the latter population.

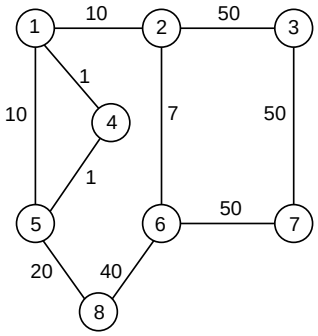
The approaches described in the previous paragraph are fundamentally different from the one proposed in this paper. One of the main differences regards the contexts of information that exist in the algorithms. Besides the endosymbiont chromosomes

that represent a short-term memory of the search, there is information in the Transgenetic Algorithms also in the host's repository and in the transgenetic vectors. These three components of the Transgenetic Algorithms are interdependent, autonomous and equivalently important for the search process. The information of the host is not necessarily encoded on chromosomes, nor necessarily represents solutions of the optimization problem being tackled. This information represents a long-term memory, not exclusively associated with the search performed by the algorithm. The information of the host's repository can be evaluated regarding the expectation of producing transgenetic vectors that are successful in manipulating the chromosomes. The transgenetic vectors are dynamic and volatile elements without a perfect match to the elements of traditional evolutionary algorithms. They cooperate with the evolution of the system host/endosymbiont, being guided by the information of the host in the task of accomplishing their transcriptions in the chromosomes.

The endosymbiont chromosomes do not reproduce or share genetic material directly. They are uniquely subjected to the pressure that results from the manipulation performed by the transgenetic vectors. The mixture of information of the host's context with those existing in the population of endosymbiont chromosomes has the potential to produce, in many cases, the diversification needed to escape from local minima. The process ends when the exchange of information between the host and the population of endosymbionts does not result in further changes (improvements) in the fitness of the endosymbionts.

4 The Hybrid Transgenetic Algorithm Applied to the PCSTP

In this section, the details of the Transgenetic Algorithm for the PCSTP are described. Let G be an undirected graph, $G = (V, E, c, w)$ and $|V| = n$. Figure 2 shows a PCSTP instance, where the weights of the vertices are listed in the table beside the graph.



v	w(v)
1	100
2	200
3	0
4	0
5	0
6	0
7	50
8	100

Figure 2: A PCSTP instance

The chromosome is an n-array in which the indices represent the nodes of the input graph, and each element is 1 or 0 whether the correspondent vertex belongs or not,

respectively, to the solution tree represented in the chromosome. The nodes of the solution tree are connected by means of a minimum spanning tree. Figure 3 shows a solution of the instance of figure 2. The fitness is given by the value of the objective function defined in equation 1. The cost of the minimum spanning tree correspondent to the chromosome of figure 3 (a) is 32. Node 7 is a demand node that is not in the solution tree. The penalty for the exclusion of node 7 is 50. Thus, the fitness of the chromosome represented in figure 3(a) is 32+50 = 82.

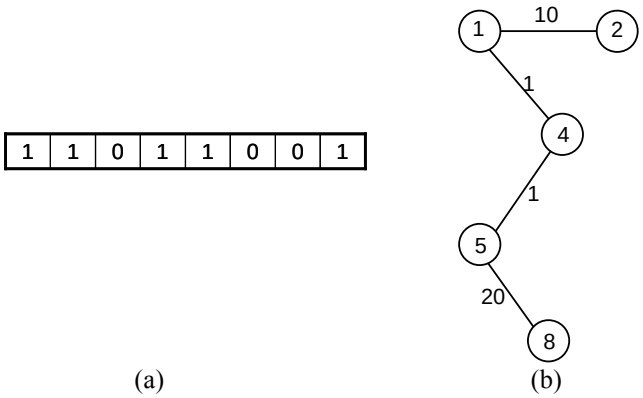
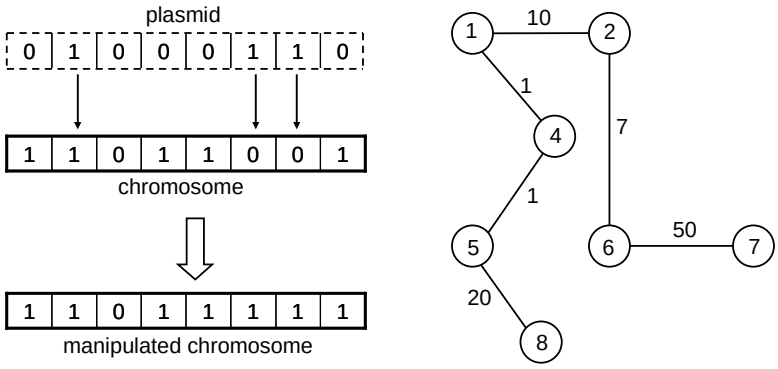


Figure 3:(a) Chromosome representing a solution of the PCSTP instance of figure 2 (b) the minimum spanning tree correspondent to the chromosome of figure 3 (a)

The information utilized by the transgenetic vectors to manipulate the chromosomes in the proposed algorithm is described in the following. *A priori* information is given by a square matrix of order *n* containing the vertices of the shortest paths between every pair of nodes. *A posteriori* information is given by the five best chromosomes found up to a given iteration. Those solutions compose a set called elite pool.

There are two types of plasmids. The information string of the first type plasmid is the shortest path between a node in the solution tree and a demand node out of the solution tree. In order to build this information string, a demand node outside the solution tree is selected at random. The shortest path between this node and any node in the solution tree is chosen to be the information string of a first type plasmid. To illustrate the first type plasmid, consider the solution tree of figure 3(b). The shortest path between node 7 and a node of the solution tree is 57. This path corresponds to go from node 7 to node 2, using the intermediary node 6. The information string of the first type plasmid for this example is formed by vertices 2, 6 and 7 (length equals 3). This plasmid is represented in figure 4(a). The transcription operator (procedure p_2) creates a clone of the chromosome that will be manipulated and copies the elements 1 from the plasmid to the correspondent positions of the clone chromosome. Figure 4 (a) shows the manipulation of the chromosome of figure 3(a). Figure 4(b) shows the minimum spanning tree that corresponds to the manipulated chromosome of figure 4(a). In this paper, procedure p_1 utilizes the fitness to evaluate the susceptibility of a given chromosome to the manipulation by a transgenetic vector. In this case, if the

fitness of the clone chromosome after the inclusion of the plasmid's string is better than the fitness of the original chromosome, then the original chromosome is susceptible to the manipulation. Therefore, the manipulated clone replaces the original chromosome in the population. In the example shown in figure 4, the manipulated chromosome with fitness 79 is better than the original one since the latter has fitness 82 and this is a minimization problem.



(a) (b)

Figure 4:(a) Chromosome manipulated by a first type plasmid (b) the minimum spanning tree correspondent to the manipulated chromosome of figure 4 (a)

A chromosome, *C*', selected at random from the elite pool is the source of information for the second type plasmid. The information string of the second type plasmid is a segment between two indices, *r* and *s*, also randomly selected, $|r-s| \leq \lfloor n/4 \rfloor$ of *C*'. The elements 1 in the selected fragment form the information string of the second type plasmid. The manipulation method utilized by the second type plasmid is the same utilized by the first type plasmid. The manipulation, however, may result on a tree with leaves that are not demand nodes. In this case, a pruning procedure removes those leaves. The pruning procedure is described further. Procedure p_1 is the same defined for the first type plasmid.

The other transgenetic vector utilized in this paper is the transposon. It rearranges the genes of a given chromosome in the locci that are determined by procedure p_3. In this work, procedure p_3 selects two indices, at random, in the target chromosome. Those indices define a segment of the target chromosome. The transposon's information string is a rule to withdraw vertices from the segment determined by procedure p_3. The removal of those vertices, operated by the transposon's procedure p_2, is done one vertex at a time. Iteratively, each element 1 in the segment is set to 0. The other elements remain the same as in the original chromosome. If leaves representing non demand nodes remain in the solution tree, they are pruned. The best resultant solution tree is considered as the manipulated clone chromosome. Procedure p_1 is the same designed for the plasmids. Let the manipulated chromosome of figure 4(a) be the original chromosome for the next example with r =2 and s = 6. Then, each element 1 of the chromosome is removed and the resultant chromosome is analyzed. Figures 5(a) and 5(b) show two resultant chromosomes with the removal of vertices 2

(figure 5 (a)) and 4 (figure 5(b)). The solution trees corresponding to those chromosomes have cost 312 and 97, respectively.

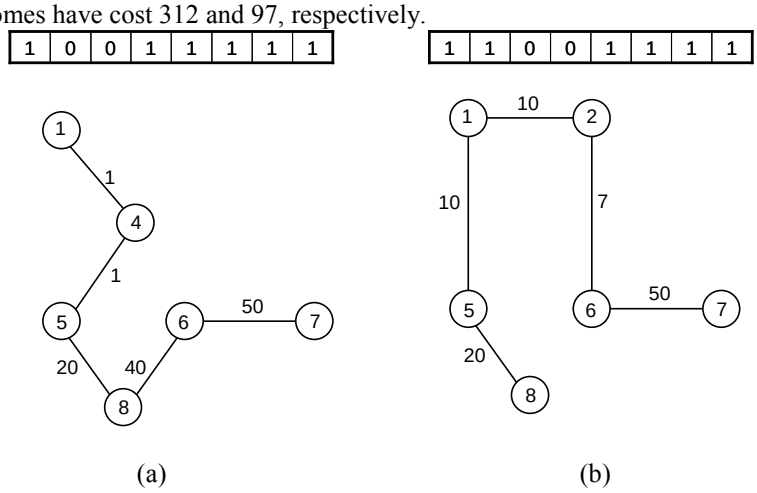


Figure 5: Chromosome manipulated by a transposon (a) with vertex 2 set to 0 and (b) with vertex 4 set to 0

The pseudo-code of the hybrid transgenetic algorithm developed for the PCSTP is shown in figure 6. Initially, a population of chromosomes is generated (step 1). Each chromosome is the result of the following process. At first, a set of nodes is obtained with an implementation of a primal-dual algorithm with processing time in $O(n^2 \log n)$ [Johnson et al. 2000]. The set of nodes returned by this algorithm induces a subgraph of the original graph. If this subgraph is connected, a minimum spanning tree is built with the nodes of the induced subgraph, otherwise the solution returned by the algorithm is discarded. To generate distinct solutions, perturbations are done in the original graph. The two methods used to disturb the graph are: vertex elimination and prize change [Canuto et al. 2001]. In the first perturbation method, each demand node of the last solution returned by the primal-dual algorithm has a probability of 50% to have their prizes set to zero in the original graph. In the second perturbation method a disturbing factor α is generated in the interval [0,1] for each demand node of the original graph. Then, the prize $w(v)$ of each demand node v is replaced by $\alpha \times w(v)$. The leaves of the minimum spanning tree obtained with the set of vertices returned by the primal-dual algorithm are scanned. If the prize of any leaf is less than the cost of the edge connecting this leaf to the tree, then the leaf is withdrawn. If new leaves are obtained after this process, then they are scanned too. The processing time of the pruning algorithm has complexity $O(n^2)$.

The host's repository is initialized with the matrix of shortest paths and the 5 best chromosomes of the initial population (step 2). Chromosome C^* is initialized with the best chromosome of the initial population (step 3). In step 4, variable β is initialized. This variable controls the search stage, storing the number of the current iteration. While a maximal number of iterations, *#generations*, is not reached, the algorithm iterates between steps 5 and 25.

```
Input: Graph G
Output: Chromosome C*
1 Generate population Pop
2 Load host's repository (HR)
3 C* ← Best chromosome of Pop
4 β ← 1
5 while (β ≤ #generations) do
6    for j ← 0 to |Pop| do
7       u ← random(#generations)
8       if (u >β) then
9          C' ← attack_plas(C_j, β)
10      else
11         C' ← attack_trans(C_j)
12      C'← path_relinking(C')
13      if C' ∈ Pop then
14         C' ← disturb(C')
15      if (f(C') < f(C*) ) then
16         C* ← C'
17         update_elite_pool_HR(C*)
18      if (f(C') < f(C_j) ) then
19         C_j ← C'
20      else
21          x ← random(100)
22          if (x ≤ 5) then C_j ← C'
23  end_for_j
24 β ← β + 1
25 end_while
```

Figure 6: Pseudo-code of the hybrid algorithm for the PCSTP

Variable u stores a pseudo-random integer, chosen with uniform probability between 1 and #*generations* (step 5). On each iteration step, a transgenetic vector is selected to manipulate the chromosomes of the whole population. If a plasmid is chosen in a given iteration, then the type of plasmid to be used has also to be defined. Variables u and β are compared and the probabilities of selecting a plasmid or a transposon, as well as the probability of selecting a plasmid of the first or second type, are established. Plasmids are more likely to be chosen at initial iterations. The same occurs with the probability of choosing a first type plasmid that is also higher than the probability of choosing a second type plasmid at initial iterations. As the algorithm runs, those probabilities are modified and, at the end, transposons and the second type plasmid are more likely to occur. This strategy is adopted to benefit the search in each stage. For instance, in early stages of the search process the quality of solutions represented in the population is weak. As a result, the elite pool in the initial iterations has, probably, low quality solutions. Thus, the information that comes from the matrix

of shortest paths is more useful at initial iterations. The quality of the elite pool tends to be improved during the search. An analogous reasoning is done concerning the choice of plasmids or transposons.

Procedure *attack_plas*() (step 9) implements the manipulation of chromosome C_j by a plasmid. Chromosome C_j and the parameter β are the input data for this procedure. The resultant chromosome is stored in *C*'. Likewise, the procedure *attack_trans*() (step 11) implements the manipulation of chromosome C_j by a transposon.

After the manipulation, the resultant chromosome is the input of a path-relinking procedure (step 12). Procedure *path_relinking*() returns the best chromosome that results from a path-relinking operation between chromosome *C*' and the closest chromosome of the 5 chromosomes in the host's database. A previously presented path-relinking procedure [Canuto et al. 2001] is adapted to be utilized in the proposed algorithm. First, the algorithm finds the solution (chromosome) *X* which is closest to the solution represented in *C*'. The Hamming distance is utilized. The vector *M* of differences between *X* and *C*' is computed. An element of *M* is 0, if the vertex corresponding to an index is in or out both solutions, 1 if the vertex is in *C*' and not in *X* or 2 if the vertex is in *X* and not in *C*'. *M* is scanned. If an element 1 (resp. 2) is found, the vertex is removed from (resp. inserted in) *C*'. A minimum spanning tree is built with the vertices of each intermediary solution that corresponds to a vertex insertion in (resp. removal from) *C*'. The best intermediary solution is returned by the procedure *path_relinking*().

In step 13 the algorithm checks if the resultant chromosome is already in the population. If this is the case, then the chromosome is disturbed (step 14). A demand node which is not in the solution represented in that chromosome is added to it and a new spanning tree is generated. Step 15 checks if the resultant chromosome is a new best current solution. If this is the case, then the resultant chromosome is copied to *C** (step 16) and the elite pool of the host's database is updated (step 17). The updating of the elite pool is done with the inclusion of the new best solution and the removal of the worst solution.

If the resultant chromosome is better than the original one, the former replaces the latter (steps 18-19). However, if the manipulated chromosome is worse than the original one, the former can still replace the latter with a probability of 5% (steps 21-22).

5 Computational Experiments

The proposed algorithm was coded in C and run on a Pentium IV, 2.8GHz, 256 Mb, with Linux, gcc compiler. The experiment was done with 114 instances of classes: K, P, C and D, available at http://www.research.att.com/ flmgcr/data. The solutions of those instances were obtained with exact algorithms [Lucena and Resende 2004, Ljubić et al. 2004]. Fifty independent runs of the proposed algorithm were executed for each instance. The size of the population is 25, the maximal number of generations is 60. An additional stopping criterion was utilized, being 20 generations without improvement of the best solution.

The performance of the hybrid transgenetic algorithm, HTA, was compared with the published results of the multi-start algorithm with Variable Neighborhood Search and Path-relinking [Canuto et al. 2001], LSP, the Memetic Algorithm [Klau et al. 2004], MA, and the Lagrangian heuristic [Cunha et al. 2008], LH. The results for each instance are presented in tables 1-3. Table 4 summarizes the results of tables 1-3, showing average results for each class and instance size. Tables 1-3 show the results of the computational experiment for groups P, K, C and D. The identification of each instance is given in the first column. Columns labeled *gap* show the percent difference from the best solution found by each algorithm and the optimal solution. Columns labeled T(s) show the average processing time in seconds.

Running time comparisons are, in general, difficult to make, even when the same codes, the same machines and the same compiler options are utilized. In this paper, this comparison is still more difficult, since the test platforms are different. The platforms of LSP, MA and LH are, respectively, Pentium II, 400 MHz, 64 Mb of RAM; Pentium IV, 2.8 GHz, 2Gb of RAM; and Pentium IV, 3.0 GHz, 512Mb of RAM. Running the tests in the same platform was not possible, once the authors of the other algorithms could not make their codes available. Those authors utilize software packages that are not available. A re-implementation of those algorithms could introduce errors and the results obtained with the new implementations could produce results that differ largely from the published ones. The platforms of MA and LH are close to the platform of HTA. The test platform of the LSP, however, is significantly inferior to the platform of the other algorithms. Then, following the procedure adopted in other papers [Klau et al. 2004], we divide the processing time of the LSP by a factor of 10, a very good estimate of LSP's processing time in machines similar to the ones utilized by the other algorithms. As well, no significance statistical tests could be done in order to compare the three algorithms.

The results for class P are shown in Table 1. The four algorithms find the optimal solution of the five instances of class P100. Only algorithm LH does not find the optimal result for instance P200. Regarding the five instances of class P400, table 1 shows that the LSP finds the optimal solutions of all instances, the MA and the HTA do not find the optimal solution of two instances and the LH does not find the optimal solution of any instance. HTA exhibits the best processing times for instances of class P, with an exception for instance P400.3 which the best processing time is obtained by the LH. In average, the HTA presents processing times 12.9, 7.2 and 1.8 times lower than the LSP, the MA and the LH, respectively.

The proposed algorithm finds the optimal solution of 22 of the 23 instances of class K shown in table 1. LSP, MA and LH fail in obtaining the optimal solution of 3, 7 and 12 instances, respectively. The proposed algorithm is approximately 34, 20 and 17 times faster than LSP, MA and LH, respectively.

Table 2 shows that the HTA and the LSP do not find the optimal solution of 1 instance of the 20 instances of class *Cxx-A*. The MA and the LH do not present optimal results for 3 and 8 instances of class *Cxx-A*, respectively. Concerning instances *Cxx-B*, the HTA does not find the optimal result of 2 instances. The LSP, the MA and the LH do not find the optimal result of 1, 6 and 15 instances, respectively. In average, the HTA is 10.5 and 1.3 times faster than the LSP and the MA, respectively, and 1.8 times slower than the LH for instances of class *Cxx-A*. Nevertheless, the HTA presents quality of solution 5.9 times, in average, better than

the LH for instances of class *Cxx-A*. In average, the HTA is 17.7, 3.8 and 1.3 times faster than the LSP, MA and LH, respectively, for instances of class *Cxx-B*.

Instance	**LSP**		**MA**		**LH**		**HTA**	
	gap	t(s)	gap	t(s)	gap	t(s)	gap	t(s)
P100	0	1.5	0	2.7	0	0.25	0	0
P100.1	0	1.4	0	3.3	0	0.32	0	0.01
P100.2	0	0.5	0	2.4	0	0.31	0	0.03
P100.3	0	1.0	0	2.8	0	0.23	0	0.03
P100.4	0	1.0	0	2.3	0	0.18	0	0.01
P200	0	7.2	0	7.2	0.45	1.58	0	0.7
P400	0	39.7	0.2	29.3	0.37	5.74	0.2	1.4
P400.1	0	38.2	0	19.3	1.14	7.52	0	1.3
P400.2	0	39.6	0	17.8	0.26	5.18	0	2.1
P400.3	0	50.0	0	23.1	0.44	5.99	0	11.6
P400.4	0	56.5	0.5	21.4	0.89	6.28	0.2	1.1
K100	0	0.2	0	1.7	0	0.11	0	0
K100.1	0	0.2	0	1.6	0	0.09	0	0
K100.2	0	0.3	0	1.6	0	0.18	0	0
K100.3	0	0.3	0	1.5	0	0.25	0	0
K100.4	0	0.6	0	1.9	0	0.04	0	0
K100.5	0	0.2	0	1.4	0	0.06	0	0
K100.6	0	0.2	0	1.2	0	0.02	0	0
K100.7	0	0.2	0	1.6	0	0.10	0	0
K100.8	0	0.2	2.3	1.5	0	0.34	0	0
K100.9	0	0.2	0	1.4	0	0.02	0	0
K100.10	0	0.1	0	1.4	0	0.03	0	0
K200	0	0.9	0	2.4	3.96	0.74	0	0
K400	0	6.8	0	6.9	7.09	2.73	0	0
K400.1	0	19.4	0	6.7	9.32	2.75	0	0
K400.2	0.2	23.4	0.2	7.1	11.73	5.57	0	1.5
K400.3	0	14.0	0	7.4	6.20	2.39	0	0
K400.4	0	20.4	0.1	7.7	6.20	2.07	0	0.8
K400.5	0	12.2	0.3	7.1	4.87	2.33	0	0.2
K400.6	0	6.0	0	8.3	6.34	3.34	0	0.1
K400.7	0.1	30.6	0.1	7.7	7.46	3.07	0	0.9
K400.8	0	4.2	0	6.7	4.94	3.42	0	0
K400.9	0	7.6	0.1	7.5	7.21	3.00	0	1
K400.10	0.4	23.1	0.7	8.7	12.73	2.76	0.2	0.4

Table 1: Results for instances of classes P and K

Instance	LSP		MA		LH		HTA	
	gap	t(s)	gap	t(s)	gap	t(s)	gap	t(s)
C01-A	0	3.0	0	1.9	0	0.18	0	0
C01-B	0	5.8	0	2.3	2.30	0.84	0	0
C02-A	0	7.0	0	1.8	0	0.18	0	0
C02-B	0	5.4	0	2.2	1.15	0.85	0	0
C03-A	0	8.7	0	2.9	0	0.50	0	0
C03-B	0	29.4	0	10	0.47	2.16	0	6
C04-A	0	14.8	0	4.5	0.16	1.60	0	7.6
C04-B	0	38.7	0.36	39	0.82	3.49	0.09	10.9
C05-A	0	44.7	0	9.4	0	1.48	0	10.6
C05-B	0	39.7	0	20.1	0.16	2.56	0	0
C06-A	0	0.9	0	4.1	0	0.21	0	0
C06-B	0	17.9	0	5.6	9.11	3.28	0	0.05
C07-A	0	3.4	0	3.5	0	0.38	0	0
C07-B	0.98	16.7	3.92	5.7	0	2.36	0.98	0.9
C08-A	0	31.3	0	8.5	0.34	4.78	0	0.5
C08-B	0	40.4	0.4	29	0.70	6.52	0	8
C09-A	0	47.5	0	13.4	0.56	6.57	0	27.4
C09-B	0	58.3	0.76	38.5	0.58	10.28	0	24.3
C10-A	0	62.8	0	35.4	0.24	5.90	0	90.3
C10-B	0	47.4	0	39.4	0.17	6.67	0	9.5
C11-A	0	12.8	0	6.1	0	1.00	0	0
C11-B	0	1.4	0	9.1	7.69	5.50	0	0.02
C12-A	0	16.2	0	9	0	2.24	0	0
C12-B	0	15.6	0	8.7	4.83	6.38	0	0
C13-A	0.42	105.0	0.42	17.9	0.55	11.08	0.84	6.8
C13-B	0	73.3	0	35.9	0.79	14.71	0	2.8
C14-A	0	82.9	0	21	1.00	13.20	0	4.2
C14-B	0	76.6	0	29.8	0.91	13.69	0	2.3
C15-A	0	95.7	0	45.4	0.58	16.49	0	15.3
C15-B	0	83.7	0	45.7	0.36	14.89	0	0.01
C16-A	0	192.0	9.09	10.6	0	5.09	0	0.7
C16-B	0	175.8	9.09	11.5	0	9.38	0	0.9
C17-A	0	54.9	5.55	11.2	0	7.84	0	0.2
C17-B	0	43.4	0	12.7	0	7.80	0	0.3
C18-A	0	399.0	0	24.1	1.523	15.52	0	9.5
C18-B	0	326.2	0	26.2	1.670	16.75	0	16.7
C19-A	0	392.8	0	17.9	0	5.58	0	7.3
C19-B	0	339.0	2.05	15.8	0	4.33	0	12.3
C20-A	0	431.1	0	7.3	0	2.88	0	8.1
C20-B	0	380.0	0	5.2	0	1.41	0	7.4

Table 2: Results for instances of class C

Instance	LSP		MA		LH		HTA	
	gap	t(s)	gap	t(s)	gap	t(s)	gap	t(s)
D01-A	0	0.6	0	3.1	0	0.75	0	0
D01-B	0	25.7	0	3.8	1.84	2.34	0	0.6
D02-A	0	0.7	0	3.5	0	0.74	0	0
D02-B	4.58	48.6	0	7.3	0	1.98	0	7.1
D03-A	0	73.4	0	7.4	0	2.75	0	34.4
D03-B	0.06	218.4	0.47	51	0.32	8.21	0.66	255.3
D04-A	0	126.3	0	10.4	0.24	5.10	0.41	99.1
D04-B	0	223.3	0.22	49.6	0.16	7.46	0.21	537.1
D05-A	0	335.2	0	29.1	0.09	8.07	0.27	669.7
D05-B	0	255.5	0.08	65.1	0.35	10.16	0.09	1200.2
D06-A	0	2.0	0	7.7	0	0.93	0	0
D06-B	4.47	70.2	8.35	10.5	8.66	11.40	1.49	8.7
D07-A	0	19.5	0	8.2	0	1.78	0	0
D07-B	1.94	71.1	1.94	9.5	2.27	11.11	0	0
D08-A	0	172.7	0	19.1	0.40	17.84	0.39	160.7
D08-B	0.19	317.5	0.93	123.8	0.42	33.46	0.19	346.1
D09-A	0.18	410.9	0.43	52.1	0.41	22.52	0.56	365.5
D09-B	0	275.4	1.15	151.2	0.32	47.73	0.07	224.8
D10-A	0	419.3	0.2	122.2	0.17	37.11	0.11	878
D10-B	0	264.4	0.51	107.3	0.20	43.25	0.04	1108.6
D11-A	0	54.0	0	15.4	0	4.82	0	0
D11-B	3.44	128.0	0	17.4	11.96	20.58	3.44	2.2
D12-A	0	84.4	0	13.9	3.73	28.64	0	0
D12-B	0	68.7	0	15.1	3.25	23.98	0	0.1
D13-A	0	504.7	0.38	58.7	0.50	58.28	0	195.4
D13-B	0	428.8	1.17	97.2	0.42	81.47	0.2	359.3
D14-A	0	638.8	0.59	102.3	0.62	86.95	0	302.9
D14-B	0	617.8	1.38	102.8	0.41	121.85	0	452.9
D15-A	0	784.0	0.64	145.7	0.29	105.53	0	274.4
D15-B	0	522.0	0.6	95.6	0.18	92.06	0	389.6
D16-A	0	139.7	7.69	23.1	0	22.83	0	0
D16-B	0	104.3	0	26.4	0	23.47	0	0
D17-A	0	350.6	0	24.8	5.16	43.90	0	0.07
D17-B	0	208.9	0	23.7	5.20	47.58	0	0.08
D18-A	0	3004.4	1.28	81.4	0.87	76.55	0.45	215.5
D18-B	0.44	3664.3	3.22	98.7	0.66	71.48	1.34	312.5
D19-A	0.65	4095.5	3.82	87.6	0.35	89.42	1.3	525.2
D19-B	0.32	3860.0	2.51	81.9	0.43	71.23	0.64	700.5
D20-A	0	2813.9	0.18	18.4	0.19	38.75	0	111.9
D20-B	0	2210.4	0	12.7	0.19	9.21	0	170.7

Table 3: Results for instances of class D

Regarding the number of optimal solutions found by each algorithm of class D, table 3 shows that the LSP exhibits the best performance obtaining the optimal solution for 30 of the 40 instances. The LSP is followed by the HTA, the MA and the LH. Those algorithms find the optimal solution of 22, 18, and 9 instances respectively. Table 4 shows that considering quality of solution, in average, the LSP obtains the best results for instances of class Dxx-A, and the HTA is the best algorithm for instances of class Dxx-B. The HTA presents worse runtimes than the LSP for 6 instances, however, in average, the former is 3.6 and 2.2 times faster than the latter for instances of classes Dxx-A and Dxx-B, respectively. Concerning quality of solution the HTA outperforms the MA and the LH in 19 and 23 instances, respectively. MA and LH present better solutions than HTA in 7 and 8 instances, respectively. In average, table 4 shows that the HTA exhibits quality of solution better than the MA and the LH for classes D-A and D-B. The quality of solutions of MA for classes D-A and D-B is improved by a factor of 4.4 and 3.7 times, respectively, by the HTA. The HTA exhibits higher processing times than the MA, in average, for instances of class D, being 4.6 and 5.3 times slower than the latter for classes D-A and D-B, respectively. The LH is faster than the HTA by a factor of 5.9 and 8.2 times for instances of classes D-A and D-B, respectively. The HTA presents better quality of solution than the LH, improving the average solutions of the latter by a factor of 3.7 and 4.4 times for classes D-A and D-B, respectively.

Instance		**LSP**		**MA**		**LH**		**HTA**	
		gap	t(s)	gap	t(s)	gap	t(s)	gap	t(s)
K	100	0	0.25	0.209	1.53	0	0.11	0	0
	200	0	0.90	0	2.40	3.96	0.74	0	0
	400	0.064	15.25	0.136	7.44	7.645	3.04	0.018	0.45
P	100	0	1.08	0	2.70	0	0.26	0	0.02
	200	0	7.20	0	7.20	0.450	1.58	0	0.70
	400	0	44.80	0.140	22.18	0.620	6.14	0.080	3.50
C	A	0.021	99.88	0.753	12.80	0.248	5.14	0.042	9.43
	B	0.049	90.74	0.829	19.62	1.586	6.69	0.054	5.12
D	A	0.042	701.53	0.761	41.71	0.651	32.66	0.175	191.64
	B	0.772	679.17	1.127	57.53	1.862	37.00	0.419	303.82

Table 4: Average results

Table 4 shows that the LSP obtains 8 best average results from the 10 classes of instances regarding quality of solution. The proposed algorithm obtains the best average results in 6 of the 10 classes. The average processing times spent by the HTA are significantly lower than the processing times of the LSP for all classes of instances. The MA and the LH do not present any average result, concerning quality of solution, better than the HTA. The MA and the LH present better average processing times than the proposed algorithm for instances of class D. The LH presents the best average processing times for instances of class *C-A*. Nevertheless, those gains in processing times result in loss of quality of solution for the MA and the LH.

6 Conclusions

In this paper a Transgenetic Algorithm hybridized with a path-relinking procedure was applied to the Prize Collecting Steiner Tree Problem. Alike bacteria that exchange precious information for survival, evolution in Transgenetic Algorithms takes place by cooperation rather than by competition. These algorithms base the search process in information obtained *a priori* and in information obtained from the search process itself. In this paper the *a priori* information was a matrix of shortest paths between all pairs of demand nodes. That information was utilized by the plasmids to alter the code of chromosomes, mainly at early stages of the search process. Another type of information, also inserted in chromosomes by the plasmids, was obtained from a very simple source of information about the search process, being the five best chromosomes found up to a given iteration. The utilization of those sources of information aimed at guiding the search to promising regions of the space of solutions. For the purpose of search intensification, a randomized local search procedure was applied to restricted parts of chromosomes. It was operated by means of transposons. Another intensification strategy is implemented by means of a path-relinking procedure. The proposed heuristic algorithm was applied to 114 benchmark instances and its results were compared with other heuristics presented previously for the investigated problem. Three algorithms were utilized for performance comparison: a multi-start algorithm with Variable Neighborhood Search and Path-relinking [Canuto et al. 2001], a Memetic Algorithm [Klau et al. 2004], and a Lagrangian heuristic [Cunha et al. 2008]. The results show that, in average, the proposed algorithm outperforms the Memetic Algorithm in 7 of the 10 instance classes utilized in the computational experiment, and outperforms the Lagrangian heuristic in 8 of the 10 classes, concerning quality of solution. The Memetic Algorithm and the Lagrangian heuristic do not present better average solutions than the proposed algorithm for any class of instances. The multi-start algorithm [Canuto et al. 2001] presents 4 average solutions better than the proposed algorithm and the latter presents 2 better average solutions than the first. The comparison between the processing times exhibited by the proposed algorithm and the multi-start algorithm shows that significant better results are achieved by the proposed algorithm.

7 Future Work

Concerning the Prize Collecting Steiner Tree Problem a number of alternatives exist for testing as sources of information of Transgenetic Algorithms which can make those algorithms still more powerful for the investigated problem concerning both processing time and quality of solution. Future works will investigate lower bounds and statistics of the search process as sources of information for the transgenetic vectors. Other related problems will also be tackled by the Transgenetic Algorithms.

Acknowledgements

We want to thank Mauricio Resende for having made available details of the results of his algorithm. This work was partially supported by the program PRH-22 of the National Agency of Petroleum (ANP) and CNPq.

References

[Balas 1989] Balas, E.: "The Prize Collecting Traveling Salesman Problem"; Networks, 19 (1989), 621–636.

[Bienstock et al. 1993] Bienstock, D., Goemans, M. X., Simchi-Levi, D., Williamson, D.: "A Note on the Prize-collecting Traveling Salesman Problem"; Mathematical Programming, 59 (1993), 413–420.

[Canuto et al. 2001] Canuto, S. Resende, M.G.C, Ribeiro, C.: "Local Search with Perturbations for the Prize Collecting Steiner Tree Problem in Graphs", Networks, 38 (2001), 50-58.

[Chan et al. 2005] Chan, T-M., Man, K-F., Tang, K-S., Kwong, S.A.:"Jumping Gene Algorithm for Multiobjective Resource Management in Wideband CDMA"; The Computer Journal, 48, 6 (2005), 749-768.

[Cunha et al. 2008] Cunha, A. S., Lucena, A., Maculan, N., Resende, M.G.C.: "A Relax and Cut Algorithm for the Prize Collecting Steiner Problems in Graphs"; Discrete Applied Mathematics, accepted for publishing, (2008).

[Dantzig et al. 1954] Dantzig, G.B., Fulkerson, D.R., Johnson, S.M.: "Solution of a Large Scale Traveling Salesman Problem"; Operations Research, 2 (1954), 393-410.

[Feofllofl et al. 2007] Feofllofl, P., Fernandes, C. E., Ferreira, C. E., Pina, J. C.: "Primal Dual Approximation Algorithms for the Prize Collecting Steiner Tree Problem", Information Processing Letters, 103, 5, (2007) 195-202.

[Glover 1994] Glover, F.: " Search for Nonlinear and Parametric Optimization (with Links to Genetic Algorithms)",),” Discrete Applied Mathematics, 49 (1994), 231-255.

[Goemans and Williamson 1996] Goemans, M. X., Williamson, D. P.: "The Primal Dual Method for Approximation Algorithms and its Application to Network Design Problems". In Hochbaum, D. (Ed.), Approximation Algorithms for NP-hard Problems (1996), 144-191.

[Goldbarg et al. 2007] Goldbarg, E.F., Goldbarg, M.C., Bagi, L.B.: "Transgenetic Algorithm: A New Evolutionary Perspective for Heuristics Design"; Proceedings of GECCO 2007 Genetic and Evolutionary Computation Conference, Workshop ENAS (2007), 2701-2708.

[Haouari et al. 2008] Haouari, M., Layeb, S.B., Sherali, H.D.: "The Prize Collecting Steiner Tree Problem: Models and Lagrangian Dual Optimization Approaches"; Computational Optimization and Applications, 40 (2008), 13-39.

[Johnson et al. 2000] Johnson, D. S., Minkoff, M., Phillips, S.: "The Prize Collecting Steiner Tree Problem: Theory and Practice"; Proceedings of 11th ACM-SIAM Symposium on Discrete Algorithms, (2000), 760-769.

[Kalu et al. 2004] Klau, W., Ljubić, I., Moser, A., Mutzel, P., Neuner, P., Pferschy, U., Raidl, G., Weiskircher, R.: "Combining a Memetic Algorithm with Integer Programming to Solve the Prize Collecting Steiner Tree Problem”, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004), Lecture Notes in Computer Science, 3102 (2004), 1304-1315.

[Kim et al. 2001] Kim, J.Y., Kim, Y., Kim, Y.K.: "An Endosymbiotic Evolutionary Algorithm for Optimization"; Applied Intelligence, 15 (2001), 117-130.

[Kim et al. 2006] Kim, Y.K., Kim, J.Y., Kim, Y.: "An Endosymbiotic Evolutionary Algorithm for the Integration of Balancing and Sequencing in Mixed-model U-lines"; European Journal of Operational Research, 168 (2006), 838-852.

[Ljubić et al. 2004] Ljubić, I., Weiskircher, R., Pferschy, U., Klau, G.W., Mutzel, P., Fischetti, M.: "Solving the Prize-collecting Steiner Tree Problem to Optimality"; Technical Report, TR-186-1-04-01, 14, Technische UniversitatWien, Institut fur Computergraphik und Algorithmen, 2004.

[Ljubić et al. 2006] Ljubić, I., Weiskircher, R., Pferschy, U., Klau, G.W., Mutzel, P., Fischetti, M.: "An Algorithmic Framework for the Exact Solution of the Prize-collecting Steiner Tree Problem"; Mathematical Programming, 105 (2006), 427-449.

[Lucena and Resende 2004] Lucena, A., Resende, M.G.C.: "Strong Lower Bounds for the Prize Collecting Steiner Tree Problem in Graphs", Discrete Applied Mathematics, 141 (2004), 277-294.

[Margulis 1991] Margulis, L.: "Symbiosis as a Source of Evolutionary Innovation: Speciation and Morphogenesis"; The MIT Press (1991).

[Margulis 2004] Margulis, L.: "Serial Endosymbiotic Theory (SET) and Composite Individuality: Transition from Bacterial to Eukaryotic Genomes"; Microbiology Today 31 (2004) 172-174.

[Margulis and Sagan 1986] Margulis, L., Sagan, D.: "Microcosmos"; Summit Books, New York (1986).

[Mitchell 1998] Mitchell, M.: "An Introduction to Genetic Algorithms"; The MIT Press (1998).

[Perales-Graván and Lahoz-Beltra 2008] Perales-Graván, C., Lahoz-Beltra, R.: "An AM Radio Receiver Designed with a Genetic Algorithm based on a Bacterial Conjugation Genetic Operator"; IEEE Transactions on Evolutionary Computation, 12, 2 (2008), 1-29.

[Pierce et al. 2003] Pierce, S.K., Massey, S.E., Hanten, J.J., Curtis, N.E.: "Horizontal Transfer of Functional Nuclear Genes between Multicellular Organisms"; Biological Bulletin, 204 (2003), 237-240.

[Segev 1987] Segev, A.: "The Node Weighted Steiner Tree Problem"; Networks, 17 (1987), 1-17.

[Sörensen 2007] Sörensen, K.: "Distance Measures Based on the Edit Distance for Permutation-Type Representations"; Journal of Heuristics, 13 (2007), 35-47.

[Timmis et al. 2004] Timmis, J.N., Ayliffe, M.A., Huang, C.Y., Martin, W.: "Endosymbiotic Gene Transfer: Organelle Genomes Forge Eukaryotic Chromosomes"; Nature Reviews Genetic, 5 (2004), 123-135.

[Yeung et al. 2008] Yeung, S-H., Ng, H-K., Man, K-F.: "Multi-criteria Design Methodology of a Dielectric Resonator Antenna with Jumping Genes Evolutionary Algorithm"; International Journal of Electronics and Communication (AEÄU), 62 (2008), 266-276.