A Parallel Distributed Genetic Algorithm for the Prize Collecting Steiner Tree Problem

Francisco Rojas*, Federico Meza†
School of Engineering, Universidad de Talca
Camino Los Niches Km. 1 s/n - Curicó - CHILE
Email: *frojas@alumnos.utalca.cl, †fmeza@utalca.cl

***Abstract*—Combinatorial optimization problems are commonly tackled through the use of metaheuristics aimed to improve the way the search space is explored (diversification) and how promising areas are exploited (intensification), in order to obtain good-quality solutions. We present a distributed genetic algorithm to solve the Prize Collecting Steiner Tree Problem, a classical combinatorial optimization problem. The proposed algorithm improves the quality of the solutions by asynchronously combining distributed populations that evolve in parallel, starting from initial heterogeneous configurations. To show the effectiveness of this approach an empirical study that considers both execution time and solution quality is presented. Results show that better solutions are reached when initial populations combine configurations with high and low fitness.**

***Keywords*-Distributed Genetic Algorithms, PCST, Prize Collecting Steiner Tree Problem, Parallel Metaheuristics.**

I. Introduction

The *Prize-Collecting Steiner Tree* Problem (PCST) is a classical combinatorial optimization problem that can be stated as follows. Given a undirected graph $G = (V, E)$, with non-negative edge costs and non-negative vertex profits, the aim is to find a subtree T of G minimizing the sum of the total cost of the edges in T while maximizing the total profit of the vertices not contained in T [1].

The PCST has been proved to be $\mathcal{NP}$-complete [2]. In practice, approximate techniques –such as metaheuristics– are used to obtain suboptimal solutions in a reasonable time. However, large-size instances could derive in excessively large execution times to obtain solutions of acceptable quality. Hence, parallelization arises as a natural approach to reduce execution time while improving the quality of the solutions.

In this work, we present a parallel genetic algorithm to solve the PCST, along with a study that shows its effectiveness in improving execution time and quality of the solutions.

Genetic Algorithms (GAs) fall in the category of Evolutionary Computation algorithms, which are inspired by nature's capability to evolve living beings well adapted to their environment. Feasible solutions are modeled as individuals that interact in an environment [3]. Thus, the population of individuals corresponds to the set of feasible solutions of a problem instance. Iteratively, a set of operators are applied to the population to generate a new generation of individuals. Crossover operators recombine existing individuals to produce new individuals. Mutation operators emulate self-adaption of individuals. Natural selection of individuals is implemented through a fitness function, so individuals with a higher fitness have a higher probability to survive to the next generation. Parallel GAs found in the literature can be classified in three categories [4]: Single-population master-slave GAs, Cellular GAs, and Multiple-population distributed GAs (dGAs). There are also hybrid architectures. This work follows the dGA model, with a unidirectional ring to accomplish communication between the islands while facilitating scalability.

We performed an empirical evaluation of the dGA and proved that parallelization improve the quality of the solutions by using an heterogeneous set of populations that evolve in parallel and interact asynchronously. A sequential implementation of the GA reached better-quality solutions when better-quality initial populations were used, that is, populations with higher *fitness*. However, the parallel dGA obtained better solutions when a mixture of high and low fitness initial populations were used for the islands. Thus, it is possible to conclude that heterogeneity of the initial populations is a key issue to guarantee an adequate exploration of the search space.

II. The Genetic Algorithm

To represent solutions (individuals) we use a binary encoding $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ of size $|V|$, where each σ_k valued 1 represents a vertex present in the solution. The cost for each solution is given to each individual through a fitness function.

The tree representing a solution is built using Algorithm 1, which is based on Prim's algorithm for finding the minimum spanning tree. In certain circumstances this algorithm could produce a disjoint graph. In that case the subtree with higher fitness is chosen.

Algorithm 1 Algorithm to create a feasible solution

Require: A graph $G = (V, E)$.
Require: Set of vertices $V_T \subseteq V$ in the solution.
  $V_{new} = \{x\}$, where $x \in V_T$ a randomly selected vertex.
  $E_{new} = \{\}$
  repeat
    Choose a edge (u, v) with less weight $u \in V_{new} \wedge v \notin V_{new} \wedge v \in V_T$. {if there's multiple edges with the same weight, one is chosen randomly.}
    $V_{new} = V_{new} + v$
    $E_{new} = E_{new} + (u, v)$
  until $V_{new} = V_T$
  return $G = (V_{new}, E_{new})$

978-1-4673-9795-7/15 $31.00 © 2015 IEEE
DOI 10.1109/CSCI.2015.67

We apply two mechanisms to create the initial population. First, a random-based procedure followed by pruning the vertices that increase the cost of the solution while not improving the total income (see Algorithm 2). Second, Dijkstra's Shortest Path Algorithm is used on every vertex with positive prize, giving as result a tree that can be pruned using Algorithm 2.

Algorithm 2 Pruning leafs that add more cost than income.

procedure $prune(v,\, T = (V, E))$
  for all For all children u of v **do**
    prune(u, G)
    if $c(u, v) > p(u)$ and $isLeaf(u)$ **then**
      $V = V - v$
      $E = E - (u, v)$ {remove the edge (u, v) and u from G}
    end if
  end for

The algorithm evolves iteratively by applying several stochastic operators –such as selection, crossover and mutation– to the current population producing a new generation of individuals. Selection chooses two individuals from the current population to be the parents of a new individual. The probability of selection is proportional to its *fitness* so the individual with the highest fitness has a higher probability of survival. Once the parents are selected, crossover and mutation are applied with a probability of p_{cross} and p_{mut}, respectively. In order to guarantee a minimal degree of diversity, an offspring is discarded if there is an identical solution in the population. Algorithm 3 illustrates the steps followed by the GA.

Algorithm 3 GA to solve the PCST Problem

```
t ← 0;
Init(P(0));
Eval(P(0));
repeat
  repeat
    I1, I2 = Select(P(t));
    if random > p_cross then
      I ← Crossover(I1, I2);
    end if
    if random > p_mut then
      I ← Mutation(I);
    end if
    if I ∉ P(t) then
      Eval(I);
      Replace(I, P(t));
    end if
  until there are k new individuals in the iteration
  t ← t + 1;
until A better solution was not found after Ω iterations
```

Crossover aims to preserve in the offsprings most of the genetic characteristics of theirs parents. We used *Two-point* crossover in our work. Given two positions that define a region inside the encoding string, an offspring that preserves the outer chromosomes from one parent and the inner chromosomes from the other is generated. Mutation is applied to the offsprings in order to include random innovation in the search process thus avoiding early convergence to a local optimum. We applied *bit-flop* mutation, randomly altering a chromosome within the code of the offspring [3].

Instance	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert V_{\text{without clients}}\rvert$	$\frac{\lvert E\rvert}{\lvert V\rvert}$
C5-A	500	625	250	1,25
C18-A	500	12500	417	25
D2-B	1000	1250	990	1,25
D8-B	1000	2000	833	2
D13-A	1000	5000	833	5
K400.5	400	1457	324	3,64
E3-A	2500	3125	2082	1,25

TABLE I: Instances used in the parameter tuning

III. The Parallel Genetic Algorithm

Our Parallel GA follows the multiple-population distributed GA model. Population is divided into several subpopulations –called islands– that exchange individuals through *migration*.

For the selection of the migrants we used an elite approach, that is, only the best individuals of the source *island* are selected to migrate. Immigrant individuals that already exist in the population are rejected in order to improve diversity.

Communication is supported by an unidirectional ring topology that exhibits high scalability. Communication is asynchronous to allow the arrival of individuals at any time. We used the *ProActive* library, a parallel computing framework supporting the distributed objects model, exhibiting a high level of portability and performance [5].

IV. Experimental Design

We used five sets of instances found in the literature:

- Johnson, *et al.*, used two sets of randomly generated instances [6]. In the first set *–Group P–* instances lack of structure. On the other hand, *Group K* consists of instances whose graphs emulate city maps [7].
- Canuto, *et al.*, created test sets derived from the well known OR-Library, called *Groups C and D* [7].
- Ljubić generated a set of 40 instances from the E group of the OR-Library [8].

We used the percentual error *–GAP–* to measure the quality of the solutions. Let $f(\text{optimal solution})$ be the cost of the known optimal solution, and let $f(\text{found solution})$ be the cost of the approximated solution found by the algorithm. The GAP for the approximated solution is defined as:

$$\text{GAP} = \frac{f(\text{found solution}) - f(\text{optimal solution})}{f(\text{optimal solution})}. \quad (1)$$

Some parameters must be tuned empirically. To avoid bias we selected representative samples from every group of instances for this adjustment.

We tested 16 configurations using two methods to generate the initial populations: random setup and using Dijkstra's algorithm. The resulting 32 configurations were tested with the 7 problem instances shown in Table I. In order to avoid noise deviations and to consider the stochastic nature of the

Group	Genetic Algorithm			Distributed Genetic Algorithm		
	%-*gap*	t[s]	σ	%-*gap*	t[s]	σ
C	2.39%	441.58	0.84%	0.96%	205.29	0.55%
D	2.95%	2258.25	0.66%	1.41%	1154.77	0.39%
K	0.71%	13.74	0.06%	0.64%	15.97	0.03%
P	2.10%	146.49	0.97%	0.91%	87.95	0.22%
E	6.26%	21194.27	0.29%	3.56%	10779.52	0.64%

TABLE II: Average results for instance groups C, D, K and P

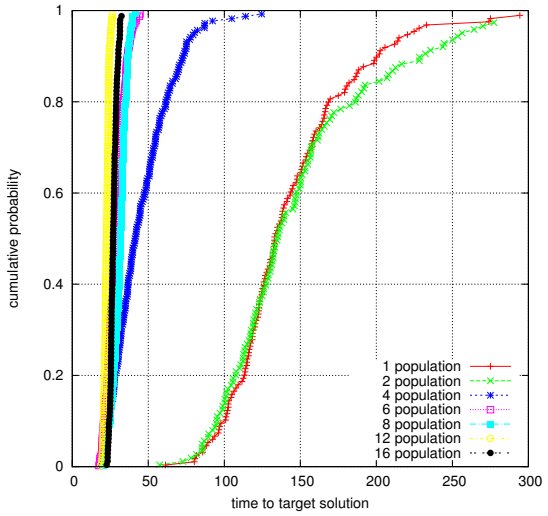

Fig. 1: Empirical time-to-target distribution obtained by the dGA for the `C13-A` instance pursuing an easy target.

algorithm, each experiment was run 5 times resulting in 1120 executions of the sequential GA to do the parameter tuning.

The most important criteria used to rank the parameter configurations was the quality of the obtained problem solution and not the execution time. The quality of a parameter configuration was computed as the sum of the best GAP and the average GAP for the 5 runs for this configuration.

The experiments were run on a 2-node/16 cores system. Each node has 2 Intel Xeon Quad Core E5430 processors running at 2.66 GHz, 8GBytes RAM, 2x6MB Cache, and an Intel 82566DM-2 Gigabit Network Interface Card. The front-end was an Intel Core 2 Duo E8300 PC running at 2.83 GHz, with 4GBytes RAM, 6 MBytes Cache, and an Intel 82566DM-2 Gigabit Network Interface Card. Both, the cluster and the front-end PC, run the Linux Rocks v5.2.2 operating system.

V. Results

Table II shows average results using instance groups C, D, K, and P. It can be seen that the dGA not only exhibits better execution time by finding solutions 50% faster –except for the K group– but it also obtains better solutions with lower standard deviation, even 76% better for the P group.

Fig. 1 shows the empirical distribution time-to-target for the `C13-A` instance for a medium-difficulty target of 267, using different population sizes. $N = 200$ independent runs were obtained for each algorithm. Configurations with 6, 8, 12, and 16 populations exhibit better performance, finding solutions in

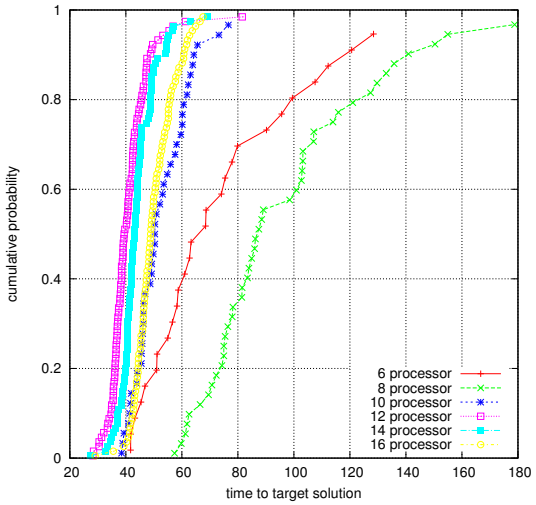

Fig. 2: Empirical time-to-target distribution obtained by the dGA for the `C13-A` instance pursuing a hard target.

less than 50 seconds with probability 1. On the other hand, the configuration with 1 population required 300 seconds.

It should be noted that not all the configurations succeeded in achieving the target solution after the 200 runs. Configurations with a higher number of populations were not only faster, but more robust in terms of finding solutions.

Fig. 2 shows the time-to-target distribution for the same instance but making the dGA pursue a more ambitious target. The dGA with 12 populations showed again the highest performance. This is probably due to the small impact on diversification and intensification associated to populations beyond the 12th, combined with a higher communication overhead introduced by additional populations.

VI. Conclusions and Future Work

The sequential GA performed better with high-quality initial populations, that is, when the individuals of the initial population had better fitness In this case, a high-quality initial population is required to produce high-quality solutions.

Results obtained with the sequential GA are encouraging, showing an average percentual error of 2.39%, 2.95%, 0.71%, 2.10%, and 6.26% for the instances C, D, K, P, and E, respectively. Please notice that no preprocessing of the graphs was performed, as it is common to find in the literature for these instances.

The behavior of the parallel dGA was significantly different. Better results were produced when a mixture of high and low quality initial populations was used, because of the positive influence that diversity introduced on the search process. In particular, the heterogeneity of the populations increased the intensification on certain regions of the search space. Also, migration of individuals between populations contributed to diversification, causing that some populations change their search focus to unexplored regions of the search space.

Scalability respect to the number of populations is an important issue for a dGA. As the number of populations grows the probability of finding better solutions is higher. However, there is a threshold for this growth. Adding more populations beyond this threshold will not improve the quality of the solutions found. This can be explained because there could be several populations exploring the same region of the search space, or searching on regions with low value in terms of the solutions produced. Besides, a higher number of populations involves a higher communication overhead resulting in higher execution times.

Results obtained with the dGA are consistent with conclusions found in other studies, in terms that the use of parallel metaheuristics based on cooperative search lead to more robust implementations [9]. The parallel dGA exhibits a significant improvement with respect to the sequential GA, producing an average porcentual error of 0.96%, 1.41%, 0.64%, 0.91%, and 3.56% for the instances C, D, K, P, and E, respectively. Execution time was also improved in the dGA, taking approximately half of the time taken by its sequential counterpart.

We intend to explore alternative encoding schemes, including a tree based encoding that appears to be more suitable for this problem. With such a scheme the solution-decoding cost could be reduced allowing variations to a solution with fast ad-hoc operations. We also plan to evaluate new procedures to generate the set of initial populations looking for higher quality solutions that could benefit both the result of the sequential GA and the heterogeneity of the populations in the dGA.

VII. Related Work

Goemans and Williamson devised a combinatorial approximation method to solve network design problems [10]. Their algorithm is based on a primal-dual schema and runs in $O(n^2 \log n)$ time $(n = V)$. They also provide an extension of their basic algorithm, for solving the unrooted PCST problem.

Other researchers improved the Goemans-Williamson algorithm by enhancing the pruning phase. Their algorithm is faster and provides solutions that are at least as good as those obtained by the original algorithm [6].

Another work presented a multi-start local-search algorithm with perturbations. Permutation are done by changing the parameters of the input graph and feasible solutions are obtained by the Goemans-Williams optimization procedure [7].

Another study presented a genetic algorithm that incorporates primal and dual information produced by Lagrangian decomposition. Their work used the same encoding used in the proposed algorithms, as well as a modified Prim's algorithm to evaluate the fitness of the individual, similar to the mechamism used by the authors [11].

Some approximated algorithms found in the literature privilege minimizing execution time over solution quality. Other authors developed a heuristic based on minimum spanning trees to solve the PCST in good overall time, but with a considerable deviation from optimum [12].

Another research group reviewed existing theoretical models to predict the effect of parameters setting on the performance of parallel genetic algorithms [13]. Meanwhile, another study explored the performance of heterogeneous island-model dGAs using random-generated configurations [14]. Their strategy exhibited a performance comparable to manually-tuned algorithms.

References

[1] I. Ljubić, R. Weiskircher, U. Pferschy, G. Klau, and P. Mutzel, "Solving the Prize-Collecting Steiner Tree Problem to Optimality," in *Proceedings of ALENEX, Seventh Workshop on Algorithm Engineering and Experiments*, 2005.

[2] I. Ljubić, R. Weiskircher, U. Pferschy, G. W. Klau, P. Mutzel, and M. Fischetti, "An Algorithmic Framework for the Exact Solution of the Prize-Collecting Steiner Tree Problem," *Math. Program.*, vol. 105, no. 2-3, pp. 427–449, 2006.

[3] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[4] G. Luque, E. Alba, and B. Dorronsoro, "Parallel Genetic Algorithms," in *Enrique Alba (Ed.). Parallel Metaheuristics: A New Class of Algorithms*. John Wiley and Sons, 2005, pp. 107–125.

[5] L. Baduel, F. Baude, D. Caromel, A. Contes, F. Huet, M. Morel, and R. Quilici, "Programming, Deploying, Composing, for the Grid," in *Grid Computing: Software Environments and Tools*, J. C. Cunha and O. F. Rana, Eds. Springer-Verlag, 2006, pp. 205–229.

[6] D. S. Johnson, M. Minkoff, and S. Phillips, "The Prize Collecting Steiner Tree Problem: Theory and Practice," in *SODA '00: Proceedings of the 11th. Annual ACM-SIAM Symposium on Discrete Algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000, pp. 760–769.

[7] S. A. Canuto, M. G. C. Resende, and C. C. Ribeiro, "Local Search with Perturbations for the Prize-Collecting Steiner Tree Problem in Graphs," *Networks*, vol. 38, p. 2001, 2001.

[8] I. Ljubić, "Exact and Memetic Algorithms for Two Network Design Problems," Ph.D. dissertation, Technische Universitt Wien, Oct. 2004.

[9] V. dat Cung, S. L. Martins, C. C. Ribeiro, and C. Roucairol, "Strategies for the Parallel Implementation of Metaheuristics," in *Essays and Surveys in Metaheuristics*. Kluwer Academic Publishers, 2002, pp. 263–308.

[10] M. X. Goemans and D. P. Williamson, "The Primal-Dual Method for Approximation Algorithms and its Application to Network Design Problems," in *Hochbaum, Dorit S. (Ed.), Approximation algorithms for NP-hard problems*. PWS Publishing Co., 1997, pp. 144–191.

[11] M. Haouari and J. C. Siala, "A Hybrid Lagrangian Genetic Algorithm for the Prize Collecting Steiner Tree Problem," *Computers & Operations Research*, vol. 33, no. 5, pp. 1274 – 1288, 2006.

[12] M. Akhmedov, I. Kwee, and R. Montemanni, "A Fast Heuristic for the Prize-Collecting Steiner Tree Problem," *Lecture Notes in Management Science*, vol. 6, pp. 207–216, 2014.

[13] E. Cantú-Paz, "Parameter Setting in Parallel Genetic Algorithms," in *Parameter Setting in Evolutionary Algorithms*, ser. Studies in Computational Intelligence, F. G. Lobo, C. F. Lima, and Z. Michalewicz, Eds. Springer Berlin Heidelberg, 2007, vol. 54, pp. 259–276.

[14] Gong, Y. and Fukunaga, A., "Distributed Island-Model Genetic Algorithms using Heterogeneous Parameter Settings," in *Evolutionary Computation (CEC), 2011 IEEE Congress on*, June 2011, pp. 820–827.