An Evolutionary Algorithm with Solution Archive for the Generalized Minimum Spanning Tree Problem

Bin Hu and Günther R. Raidl

Vienna University of Technology
Favoritenstraße 9–11/186-1
1040 Vienna, Austria
{hu|raidl}@ads.tuwien.ac.at

Abstract. We propose a concept of enhancing an evolutionary algorithm (EA) with a complete solution archive that stores evaluated solutions during the optimization in a trie in order to detect duplicates and to efficiently convert them into yet unconsidered solutions. As an application we consider the generalized minimum spanning tree problem where we are given a graph with nodes partitioned into clusters and exactly one node from each cluster must be connected. For this problem there exist two compact solution representations that can be efficiently decoded, and we use them jointly in our EA. The solution archive contains two tries – each is based on one representation, respectively. We show that these two tries complement each other well. Test results on TSPlib instances document the strength of this concept and that it can match up with the leading state-of-the-art metaheuristic approaches from the literature.

Keywords: evolutionary algorithm, solution archive, network design

1 Introduction

The evolutionary algorithm (EA) is a popular metaheuristic for solving difficult combinatorial optimization problems (COPs). When adequately applied, EAs are often able to find good approximate solutions within a huge search space in relatively short computation times. However, a common drawback is that they usually do not keep track of the search history, and already evaluated solutions are often revisited. When the selection pressure is rather high, the population size only moderate, or the mutation and recombination operators do not introduce much innovation, the population's diversity drops strongly and the EA gets stuck by creating almost only duplicates of a small set of leading candidate solutions, called "super-individuals". In such a situation of premature convergence, the heuristic search obviously does not perform well anymore, and something must be changed in the setup.

Instead of attempting to re-configure the EA until it reaches the desired performance, we propose a method for detecting already evaluated candidate

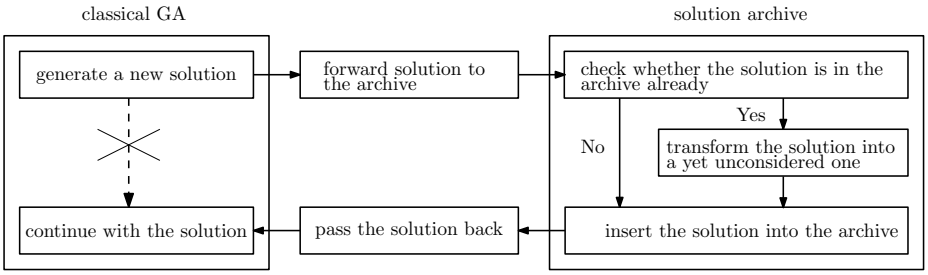

Fig. 1. Cooperation between GA and trie.

solutions and efficiently transforming them into similar but yet unvisited solutions, i.e., performing an "intelligent mutation". This is done by attaching a *complete solution archive* to the EA that stores already considered solutions in an appropriate data structure, allowing a fast detection of duplicates and an efficient conversion into unvisited solutions. In this work we realize the archive by a *trie* [2], an ordered tree data structure. Tries are typically used for effectively storing and searching large amounts of strings, e.g., in language dictionary applications. Main advantages are that the memory effort is rather low and that the costs for insertion and search operators essentially only depend on the word lengths, but not on the number of strings in the trie. Figure 1 illustrates the cooperation between the GA and the archive.

This concept has already been successfully applied on two problems where solutions are encoded as binary strings [10]. Similar methods exist where solutions are cached by hash tables [7] or stored in binary trees [14]. However, these approaches either do not support efficient conversion of duplicates or they are applied to problems with rather simple solution representations.

Here we apply the archive-enhanced EA to the generalized minimum spanning tree problem (GMSTP) which is defined as follows: Given an undirected weighted complete graph $G = \langle V, E, c\rangle$ with node set V partitioned into r pairwise disjoint clusters $V_1, \ldots, V_r$, edge set E and edge cost function $c : E \to \mathbb{R}^+$, a solution $S = \langle P, T\rangle$ is defined as $P = \{p_1, \ldots, p_r\} \subseteq V$ containing exactly one node from each cluster, i.e., $p_i \in V_i,\ i = 1, \ldots, r$, and $T \subseteq E$ being a spanning tree on the nodes in P. The costs of S are the total edge costs, i.e., $C(T) = \sum_{(u,v)\in T} c(u, v)$ and the objective is to identify a solution with minimum costs. The GMSTP was introduced in [8] and has been proven to be NP-hard. In recent years, many successful metaheuristic approaches [3–6] were developed for this problem.

2 Evolutionary Algorithm for the GMSTP

We use a classic steady state EA where the archive is consulted each time after a new solution is generated by crossover and mutation. In the following we describe the EA components.

2.1 Solution Encodings

The GMSTP has a dual-representation, i.e., two solution representations which complement each other are used together. On the one hand, the *Spanned Nodes Representation* (SNR) characterizes solutions by their set of spanned nodes P. Decoding a solution means to find a classical minimum spanning tree on P which can be done in polynomial time. On the other hand, the *Global Structure Representation* (GSR) characterizes solutions by the so-called global tree structure T^{g} where $T^{\mathrm{g}} \subseteq V^{\mathrm{g}} \times V^{\mathrm{g}}$ and $V^{\mathrm{g}} = \{V_1, \ldots, V_r\}$. It defines which clusters are adjacent in the solution without specifying the actually spanned nodes. For decoding, the optimal spanned node from each cluster can be obtained via dynamic programming in polynomial time [9]. Since T^{g} always describe a tree structure between the clusters, we store for each cluster its predecessor in the vector $\mathit{\Pi} = \{\pi_2, \ldots, \pi_r\}$ when rooting the tree at V_1.

2.2 Genetic Operators

As selection we use tournament selection of size 2. Crossover and mutation operators are implemented for both representations separately. For SNR, uniform crossover and one-point-mutation are applied on P. For GSR, edge recombination for spanning trees [11] and mutation by exchanging single global connections are implemented. Each time a new offspring is to be created, we decide randomly which representation to use.

2.3 Solution Archives

The solution archive is implemented by two indexed tries [2], storing solutions for each representation, respectively. Each trie is able to identify duplicate solutions in its associated solution encoding.

Trie based on SNR: The trie T_{SNR} is based on the vector of spanned nodes $P = \{p_1, \ldots, p_r\}$ and has maximal height r. Each trie-node at level $i = 1, \ldots, r$ corresponds to cluster V_i and contains entries $next[j]$, $j = 1, \ldots, |V_i|$. They either contain references to trie-nodes on the next level, the *complete*-flag, or the *empty*-flag. The empty-flag '/' means that none of the solutions in the subtree that would start at this point has been considered yet, while the complete-flag 'C' indicates that all solutions in the subtree have already been visited in the EA. When inserting a solution, we follow in each level i the entry that corresponds to the value of p_i. In the trie-node of the last level, $next[p_r]$ is set to 'C', indicating

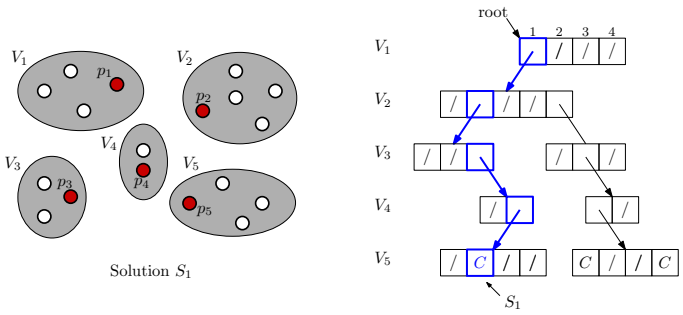

Fig. 2. Example of how solution S_1 is stored in trie T_{SNR} containing three solutions. The bold path marks the way of inserting or searching S_1.

the presence of the solution at this point. Figure 2 shows an example of how the solution S_1 is stored in T_{SNR} containing three solutions. Since we want to keep the trie as compact as possible, subtries where all solutions have been visited are pruned. This is done by removing trie-nodes that only contain C-flags and changing the entry in the previous level that pointed towards it into a C-flag.

The central feature of the solution archive is to convert duplicates upon detection. When the solution $P = \{p_1, \ldots, p_r\}$ would be revisited, it is efficiently converted into a yet unconsidered candidate solution P'. The basic idea of conversion is to backtrack to a previous trie-node on the path to the root that contains at least one yet unconsidered solution. In that trie-node on level i, $i = 1, \ldots, r$ we randomly choose an alternative entry not marked as complete and go down this subtrie following the remaining data $\{p_{i+1}, \ldots, p_r\}$ whenever possible, i.e., unless we encounter a C-flag in which case we choose again an alternative branch that must contain at least one unconsidered solution.

During conversion, biasing can be a major problem: Since the solution vector is considered in a particular order to build up the trie, some positions are subject to changes more frequently than others. Figure 3 shows an example of two conversion strategies:

- Converting the solution at the lowest possible level saves memory since it is possible to prune the trie due to completely searched subtries more often. However, when modifications at lower levels happen at higher frequency, biasing occurs.
- Converting at a random level introduces a substantially lower biasing. A drawback, however, is that not only is pruning possible less frequently, but the trie is often expanded during such an operation, resulting in a larger trie that consumes more memory.

Searching, inserting and converting solutions requires $O(r)$ time, thus the complexity only depends on the length of the solution vector, but not on the number of already stored solutions.

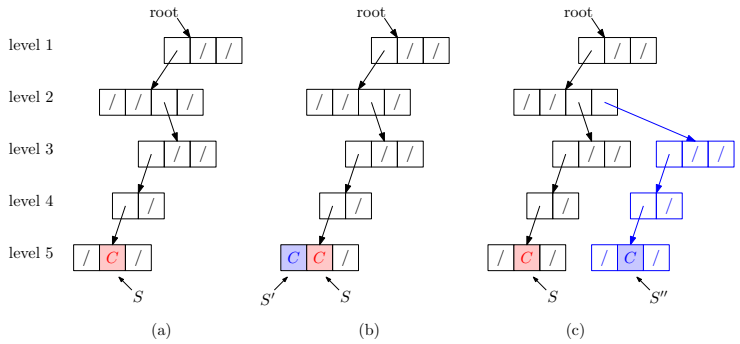

Fig. 3. a) Duplicate $P = \{1, 3, 1, 1, 2\}$ is detected. b) Conversion at the lowest level yields $P' = \{1, 3, 1, 1, 1\}$. c) Conversion at a random level (here at level 2) yields $P'' = \{1, 4, 1, 1, 2\}$.

Trie based on GSR: The trie T_{GSR} is based on the predecessors vector $\Pi = \{\pi_2, \ldots, \pi_r\}$ and has maximal height $r-1$. Each trie-node at level $i = 1, \ldots, r-1$ corresponds to the predecessor π_{i+1} and contains entries $next[j]$, $j = 1, \ldots, r$. Figure 4 shows an example of how the solution S_1 is stored in T_{GSR} containing four solutions.

Inserting, searching and converting a solution in this trie follows the same scheme as for T_{SNR}. While the first two operators require $O(r)$ time, the complexity of conversion is $O(r^2)$. This is due to the difficulties when modifying the predecessor vector. Changing one value of Π may result in an invalid tree structure. Therefore an additional repair-mechanism based on depth-first-search is required to ensure validity. Due to the larger trie-nodes, T_{GSR} is in general substantially larger than T_{SNR}.

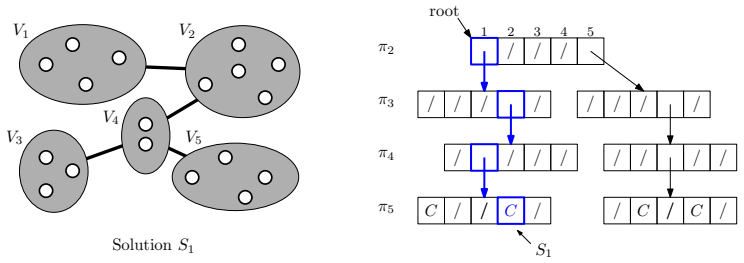

Fig. 4. Example of how solution S_1 is stored in trie T_{GSR} containing four solutions. The bold path marks the way of inserting or searching S_1.

Interaction between the tries: Since the archive consists of two tries, it is possible that a new solution created by one trie becomes a duplicate in the other trie. Therefore the conversion procedures are carried out in turn by the two tries and the solution is re-checked in the opposite trie until the derived solution is new to both tries.

3 Computational Results

We tested our approach on TSPlib[1] instances with up to 442 nodes partitioned into 89 clusters using geographical center clustering [1]. For each instance we performed 30 independent runs and each run was terminated when a time limit was reached. Standard settings and a population size of 100 were used. The EA was tested in four variants: EA without archive, EA with SNR archive based on trie T_{SNR}, EA with GSR archive based on trie T_{GSR}, and EA with full archive using both tries.

First we show in Table 1 the results obtained when terminating the algorithms after a fixed amount of time. The first two columns list the instance names (the last three digits indicate the number of nodes) and the time limit. For each EA variant we show the average final solution values $\overline{C(T)}$ and corresponding standard deviations (sd). Best results are marked bold. We observe that the EA without archive performs worst in general. Among the two variants where the archive only uses one representation, GSR is more often the better choice. If we combine both of them, the EA performs best on all instances. This

[1] http://elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsplib.html

Table 1. Results of different EA variants

		no archive		SNR archive		GSR archive		full archive	
Instance	time	$\overline{C(T)}$	sd	$\overline{C(T)}$	sd	$\overline{C(T)}$	sd	$\overline{C(T)}$	sd
gr137	150s	329.4	0.5	329.3	0.5	**329.0**	0.0	**329.0**	0.0
kroa150	150s	9830.6	31.4	9831.3	30.1	**9815.0**	0.0	**9815.0**	0.0
d198	300s	7055.1	8.7	7059.6	9.0	7044.6	2.3	**7044.0**	0.0
krob200	300s	11275.0	45.6	11248.9	7.5	**11244.0**	0.0	**11244.0**	0.0
gr202	300s	242.1	0.3	242.2	0.4	**242.0**	0.2	**242.0**	0.0
ts225	300s	62290.8	40.4	62299.1	50.9	62268.6	0.5	**62268.4**	0.5
pr226	300s	**55515.0**	0.0	**55515.0**	0.0	**55515.0**	0.0	**55515.0**	0.0
gil262	450s	945.5	4.0	945.0	3.7	942.4	2.0	**942.0**	0.0
pr264	450s	21893.2	7.7	21898.4	20.9	**21886.0**	0.0	**21886.0**	0.0
pr299	450s	20352.1	37.4	20349.7	24.9	20318.5	11.3	**20318.1**	11.3
lin318	600s	18545.9	29.2	18547.3	25.6	18525.8	12.4	**18511.0**	10.8
rd400	600s	5953.0	15.4	5959.4	20.2	5946.4	10.8	**5940.2**	6.5
fl417	600s	**7982.0**	0.0	**7982.0**	0.0	**7982.0**	0.0	**7982.0**	0.0
gr431	600s	1034.1	1.4	1033.4	0.9	1033.3	0.7	**1033.0**	0.0
pr439	600s	51921.4	60.7	51888.5	56.3	51810.5	26.5	**51791.0**	0.0
pcb442	600s	19717.0	59.5	19708.1	70.2	19632.6	21.1	**19623.7**	15.9

clearly indicates that the solution archive has a positive effect on the performance of the EA. The time overhead caused by the archive is taken into account in these results since the same time limit is used for each EA variant. The memory overhead depends on the size of the instance and ranges from around 14MB to 43MB for the SNR archive, 140MB to 480MB for the GSR archive, and 320MB to 820MB for the full archive. Note that the full archive requires substantially more memory due to the interaction between both tries.

In Table 2 we compare our EA using the full archive with several leading state-of-the-art approaches from literature consisting of a tabu search approach by Ghosh [3], a hybrid variable neighborhood search approach by Hu et al. [5], and an algorithm based on dynamic candidates sets by Jiang and Chen [6]. We observe that the proposed EA with solution archive can compete well with the other approaches. Especially on larger instances, it performs considerably well.

Table 2. Comparison with other state-of-the-art approaches

		TS	VNS		DCS		EA + archive	
Instance	time	C(T)	$\overline{C(T)}$	sd	$\overline{C(T)}$	sd	$\overline{C(T)}$	sd
gr137	150s	**329.0**	**329.0**	0.00	**329.0**	0.00	**329.0**	0.0
kroa150	150s	**9815.0**	**9815.0**	0.00	**9815.0**	0.00	**9815.0**	0.0
d198	300s	7062.0	**7044.0**	0.00	**7044.0**	0.00	**7044.0**	0.0
krob200	300s	11245.0	**11244.0**	0.00	**11244.0**	0.00	**11244.0**	0.0
gr202	300s	**242.0**	**242.0**	0.00	**242.0**	0.00	**242.0**	0.0
ts225	300s	62366.0	62268.5	0.51	**62268.3**	0.48	62268.4	0.5
pr226	300s	**55515.0**	**55515.0**	0.00	**55515.0**	0.00	**55515.0**	0.0
gil262	450s	**942.0**	942.3	1.02	**942.0**	0.00	**942.0**	0.0
pr264	450s	**21886.0**	21886.5	1.78	**21886.0**	0.00	**21886.0**	0.0
pr299	450s	20339.0	20322.6	14.67	**20317.4**	1.52	20318.1	11.3
lin318	600s	18521.0	**18506.8**	11.58	18513.6	7.82	18511.0	10.8
rd400	600s	5943.0	5943.6	9.69	5941.5	9.91	**5940.2**	6.5
fl417	600s	7990.0	**7982.0**	0.00	7982.7	0.47	**7982.0**	0.0
gr431	600s	1034.0	**1033.0**	0.18	**1033.0**	0.00	**1033.0**	0.0
pr439	600s	51852.0	51847.9	40.92	51833.8	36.07	**51791.0**	0.0
pcb442	600s	**19621.0**	19702.8	52.11	19662.5	39.79	19623.7	15.9

4 Conclusions and Future Work

In this paper we proposed for the GMSTP an EA with solution archive based on a dual-representation. The results clearly indicate that the archive improves the search performance of the EA. Considering both solution representations is also a crucial step towards overall success.

For future work, we want to investigate bounding strategies for detecting trie-branches with inferior solutions by estimating lower bounds for incomplete solutions. These branches can be pruned in order to focus the search on more

promising regions and to limit the memory overhead. We believe that the concept of solution archives is a powerful addition to EAs when it is implemented adequately for appropriate combinatorial optimization problems. Hence we want to further pursue this concept for other problems.

Acknowledgements

We thank Markus Wolf and Mika Sonnleitner, who helped in the implementation of the described concepts and did the testing as part of their master thesis [13, 12]. This work is further supported by the Austrian Science Fund (FWF) under contract nr. P20342-N13.

References

1. C. Feremans. *Generalized Spanning Trees and Extensions*. PhD thesis, Universite Libre de Bruxelles, 2001.
2. E. Fredkin. Trie memory. *Communications of the ACM*, 3:490–499, 1960.
3. D. Ghosh. Solving medium to large sized Euclidean generalized minimum spanning tree problems. Technical Report NEP-CMP-2003-09-28, Indian Institute of Management, Research and Publication Department, Ahmedabad, India, 2003.
4. B. Golden, S. Raghavan, and D. Stanojevic. Heuristic search for the generalized minimum spanning tree problem. *INFORMS Journal on Computing*, 17(3):290–304, 2005.
5. B. Hu, M. Leitner, and G. R. Raidl. Combining variable neighborhood search with integer linear programming for the generalized minimum spanning tree problem. *Journal of Heuristics*, 14(5):473–499, 2008.
6. H. Jiang and Y. Chen. An efficient algorithm for generalized minimum spanning tree problem. In *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 217–224, New York, NY, USA, 2010. ACM.
7. J. Kratica. Improving performances of the genetic algorithm by caching. *Computers and Artificial Intelligence*, 18(3):271–283, 1999.
8. Y. S. Myung, C. H. Lee, and D. W. Tcha. On the generalized minimum spanning tree problem. *Networks*, 26:231–241, 1995.
9. P. C. Pop. *The Generalized Minimum Spanning Tree Problem*. PhD thesis, University of Twente, The Netherlands, 2002.
10. G. R. Raidl and B. Hu. Enhancing genetic algorithms by a trie-based complete solution archive. In *Evolutionary Computation in Combinatorial Optimisation – EvoCOP 2010*, volume 6022 of *LNCS*, pages 239–251. Springer, 2010.
11. G. R. Raidl and B. A. Julstrom. Edge-sets: An effective evolutionary coding of spanning trees. *IEEE Transactions on Evolutionary Computation*, 7(3), 2003.
12. M. Sonnleitner. Ein neues Lösungsarchiv für das Generalized Minimum Spanning Tree-Problem. Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, 2010.
13. M. Wolf. Ein Lösungsarchiv-unterstützter evolutionärer Algorithmus für das Generalized Minimum Spanning Tree-Problem. Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, 2009.
14. S. Y. Yuen and C. K. Chow. A non-revisiting genetic algorithm. In *IEEE Congress on Evolutionary Computation (CEC 2007)*, pages 4583–4590. IEEE Press, 2007.