An Evolutionary Algorithm with Solution Archives and Bounding Extension for the Generalized Minimum Spanning Tree Problem

Bin Hu and Günther R. Raidl
Institute of Computer Graphics and Algorithms
Vienna University of Technology
Favoritenstrasse 9-11
1040 Vienna, Austria
{hu|raidl}@ads.tuwien.ac.at

ABSTRACT

We consider the recently proposed concept of enhancing an evolutionary algorithm (EA) with a complete solution archive. It stores evaluated solutions during the optimization in order to detect duplicates and to efficiently transform them into yet unconsidered solutions. For this approach we introduce the so-called bounding extension in order to identify and prune branches in the trie-based archive which only contain inferior solutions. This extension enables the EA to concentrate the search on promising areas of the solution space. Similarly to the classical branch-and-bound technique, bounds are obtained via primal and dual heuristics. 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 in the cheapest way. As the EA uses operators based on two dual representations, we exploit two corresponding tries that complement each other. Test results on TSPlib instances document the strength of this concept and that it can compete with the leading metaheuristics for this problem in the literature.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*; G.1.6 [**Numerical Analysis**]: Optimization

General Terms

Algorithms

Keywords

evolutionary algorithm, solution archive, network design, branch-and-bound

GECCO'12, July 7-11, 2012, Philadelphia, Pennsylvania, USA.

1. INTRODUCTION

Lots of complex combinatorial optimization problems (COPs) are nowadays approached by (hybrid) evolutionary algorithms (EA), which belong to the family of population based metaheuristics. In contrast to exact algorithms, they frequently scale better with increasing problem size and are able to find good approximate solutions 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 genetic operators do not introduce much innovation, the population's diversity drops strongly and in the extreme case 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 obviously does not perform well anymore. There are several established approaches for handling this problem, such as duplicate elimination or population management. These are well-known techniques for maintaining a necessary degree of diversity in the current population.

We go one step further and investigate the recently introduced *complete solution archive* as a more powerful extension for EAs that not only considers the current population, but also detects already evaluated candidate solutions over the whole search history and efficiently transforms them into similar but yet unvisited solutions by means of an "intelligent mutation". Figure 1 illustrates the cooperation between the EA and the archive.

This concept has been successfully applied to some benchmark problems where solutions are encoded as binary strings [18, 14]. In our preliminary work [9], it was applied to a complex network design problem for the first time. Now we further extend the functionality of the solution archive by introducing the so-called *bounding extension* for detecting branches in the archive which only contain inferior solutions. These branches can be pruned similarly to classical branch-and-bound (B&B) algorithms in order to focus the search on more promising regions and to limit the memory overhead. For storing already considered solutions we use the *trie* data structure [4] that allows a fast duplicate-detection and an efficient transformation into unvisited solutions. Tries are typically used for effectively storing and searching large amounts of strings, e.g., in language dic-

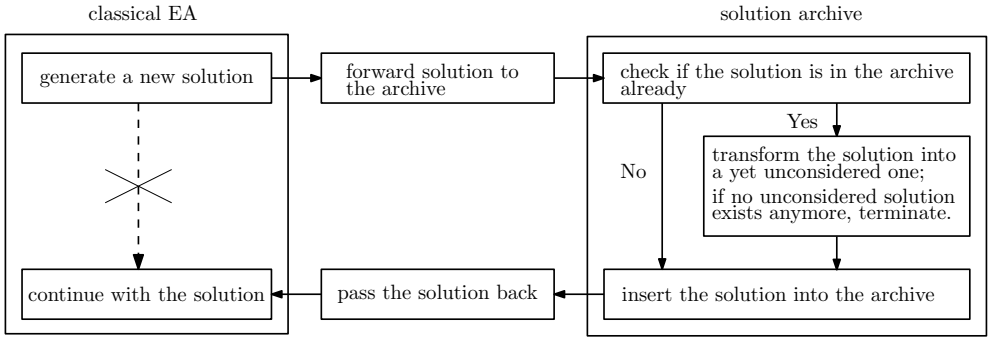


Figure 1: Cooperation between EA and trie.

tionary applications. Main advantages are that the memory effort is relatively low and that the costs for insertion and search operators essentially only depend on the word lengths, but not on the number of strings stored in the trie.

1.1 The Generalized Minimum Spanning Tree Problem

We apply our 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, see Figure 2. 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 [11] and has been proven to be NP-hard. Beside our previous work [9], many successful exact methods [2, 13, 3] and metaheuristic approaches [5, 6, 8, 10, 12] were developed for this problem in the recent years. There are several real world applications of the GMSTP, e.g., in the design of backbones in large communication networks. Devices belonging to the same existing local area network correspond to nodes within the same cluster, and the backbone is required to connect one device per local network.

From the point of view of the EA, P is encoded as a vector. Therefore we use the notation $P = \langle p_1, \ldots, p_r\rangle$ in the remaining sections.

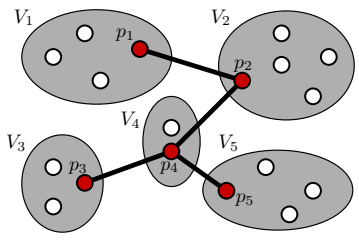


Figure 2: Example for a solution to the GMSTP.

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

Following [9], we utilize a dual-representation, i.e., two incomplete solution representations which complement each other are used together. On the one hand, the *Spanned Nodes Representation* (SNR) characterizes a solution by its spanned nodes P. Decoding such a genotype means to find a classical minimum spanning tree (MST) 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 $V^{\mathrm{g}} = \langle V_1, \ldots, V_r\rangle$. It defines which clusters are adjacent in the solution without specifying the actually spanned nodes. A decoding procedure calculates the optimal spanned node for each cluster via dynamic programming in $O(|V|^2)$ time [13]. Since T^{g} always describes a tree structure between the clusters, we store for each cluster its predecessor in the vector $\Pi = \langle \pi_2, \ldots, \pi_r\rangle$ when rooting the tree at V_1.

2.2 Genetic Operators

As selection we use tournament selection of size 2. Crossover is implemented for both representations separately. For SNR we apply uniform crossover on P and edge recombination [15] on the global tree structure is used for GSR. Each time a new offspring is created, we decide randomly which representation to use. Mutation is based on the same considerations, i.e., depending on the representation, we either exchange the spanned node in a randomly chosen cluster in SNR or a global connection in GSR. In fact, mutation is not absolutely necessary since duplicate solutions are transformed in the solution archive anyway. However, during our tests it turned out to be beneficial to include a fast and simple mutation with some low probability so that less duplicates arise and we do not need to transform so often.

3. SOLUTION ARCHIVES FOR THE GMSTP

The solution archive is implemented by two indexed tries [4], storing solutions for each representation, respectively.

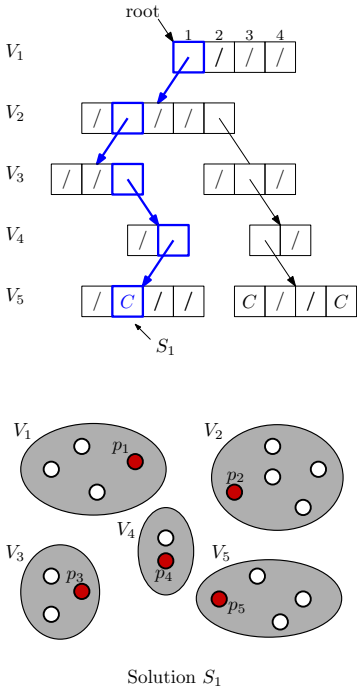


Figure 3: Example of how solution S_1 is stored in trie T_{SNR} already containing two solutions. The bold path marks the way of inserting or searching S_1.

Each trie is able to identify duplicate solutions and transforms them into new solutions in its associated solution encoding. We have to be aware though that it is possible that a new solution created by one trie becomes a duplicate in the other trie. Therefore the transformation procedures must be carried out in turn by the two tries so that a newly derived solution is re-checked in the opposite trie until it is new w.r.t. both tries. The specific implementations for the tries have been described in the previous work [9], but we summarize the essential parts in order to better understand the additional features in this work, namely the bounding extensions.

3.1 Trie Based on SNR

The trie T_{SNR} is based on the vector of spanned nodes $P = \langle p_1, \ldots, p_r \rangle$ and has maximal height r. Each trie-node at level $i = 1, \ldots, r$ corresponds to cluster V_i and contains entries $next[j]$, $j = 1, \ldots, |V_i|$, each being either a reference to a trie-node on the next level, a *complete*-flag, or an *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 by 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 the presence of the solution at this point. Figure 3 shows an example of how a solution S_1 is stored in T_{SNR}. 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.

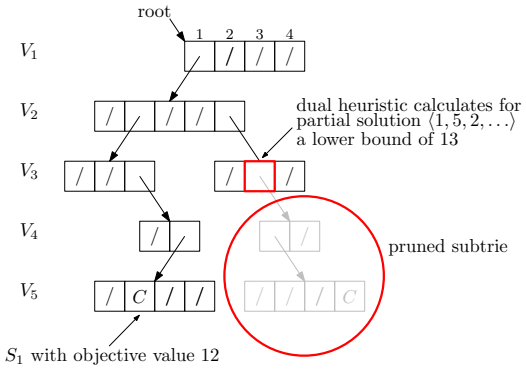
One essential feature of the solution archive is to transform duplicates upon detection. When the solution $P = \langle p_1, \ldots, p_r \rangle$ would be revisited, it is efficiently transformed into a yet unconsidered candidate solution P'. The basic idea of transformation is to backtrack to a previous 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 $\langle p_{i+1}, \ldots, p_r \rangle$ whenever possible, i.e., unless we encounter a C-flag in which case we choose an alternative branch again that must contain at least one unconsidered solution. We follow a randomized transformation strategy [9] in order to avoid a strong biasing, i.e., that some positions are subject to changes more frequently than others.

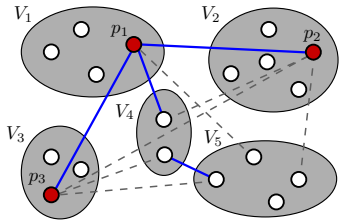
We now add the new bounding extension as an additional feature based on the following idea: Since the solution archive covers the entire solution space, choosing a branch at a particular trie-node corresponds to choosing a subspace. If reasonable bounds can be computed in these nodes for the corresponding solutions' objective values, we are able to prune whole subtries that only contain inferior solutions without explicitly considering each of them. Note that this corresponds to the underlying idea of classical B&B where primal and dual heuristics are used to compute lower and upper bounds in each B&B node, aiming at pruning subtrees that exclusively consist of invalid and/or suboptimal solutions.

For calculating a lower bound at a particular entry in a trie-node corresponding to cluster V_x, we consider the graph $G^{\text{SNR}} = \langle V^{\text{SNR}}, E^{\text{SNR}} \rangle$ which is defined as follows. V^{SNR} is composed of two sets of clusters V^{f} and V^{o}. The fixed set V^{f} consists of clusters from the trie-root to cluster V_x where the spanned nodes are fixed. We denote by $p(V_i)$, $V_i \in V^{\text{f}}$, the spanned node of cluster V_i. The open set V^{o} consists of the remaining clusters that have no fixed nodes yet. For these clusters the condition to connect exactly one node will be relaxed by allowing arbitrary edges to any, even multiple, of its nodes. E^{SNR} is composed of three sets of connections E^{ff}, E^{fo}, and E^{oo}. Set E^{ff} contains connections between clusters of set V^{f}, i.e., $E^{\text{ff}} = \{(V_i, V_j) \mid V_i, V_j \in V^{\text{f}}\}$. The costs of such a connection corresponds to the actual edge cost $c(p(V_i), p(V_j))$. Set E^{fo} contains connections between a cluster of set V^{f} and a cluster of set V^{o}, i.e., $E^{\text{fo}} = \{(V_i, V_j) \mid V_i \in V^{\text{f}},\ V_j \in V^{\text{o}}\}$. The costs of such a connection is defined as $\min\{c(p(V_i), v) \mid v \in V_j\}$. Finally, E^{oo} contains connections between clusters of set V^{o}, i.e., $E^{\text{oo}} = \{(V_i, V_j) \mid V_i, V_j \in V^{\text{o}}\}$. The costs of such a connection is defined as $\min\{c(u, v) \mid u \in V_i,\ v \in V_j\}$. Now the lower bound is obtained by applying a greedy MST algorithm on G^{SNR}.

Figure 4 shows an example for this procedure. Assume that a previously obtained solution $S_1 = \langle 1, 2, 3, 2, 2 \rangle$ has objective value $C(S_1) = 12$. A lower bound for all solutions starting with $\langle 1, 5, 2, \ldots \rangle$ is calculated on the graph G^{SNR}. The dashed lines represent the whole set of considered connections in E^{SNR} and the bold lines are the actually selected ones by the MST algorithm. Note that neither does only one node has to be connected per cluster, nor does the structure has to be connected. Assume the heuristic obtains a lower bound of 13, then there is no need to consider any further solutions in the corresponding subtrie and it is pruned.

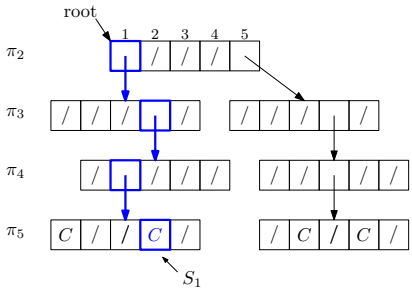
Building up E^{SNR} can be computationally expensive if it is done every time from scratch when calculating a lower

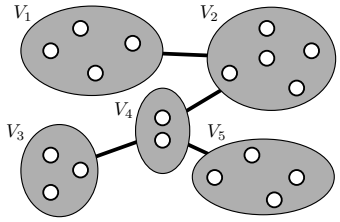




Graph G^{SNR} for determining a lower bound for partial solution $\langle 1,5,2,\ldots\rangle$

Figure 4: Example of how a lower bound is obtained for a partial solution in SNR and the corresponding subtrie is pruned due to the bound being larger than the objective value of a known solution (S_1).





Solution S_1

Figure 5: 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.

bound. Fortunately, the sets E^{fo} and E^{oo} for all possible solutions can be computed once in advance in $O(|V|^2)$ time during preprocessing. Therefore the time-complexity of the bound calculation for a partial solution is dominated by applying the MST algorithm on E^{SNR} which requires $O(r^2 \log r)$ time. Nonetheless it is not convenient to do it at each level when inserting a new solution or transforming a duplicate. On the one hand, being able to prune a subtrie at high levels means that more inferior solutions are excluded at once, thus more space and time may be saved. On the other hand, the lower bounds in these situations with relatively few fixed clusters usually are not tight enough to let this happen. In our experiments there was not a single case where such a pruning could be performed in the upper half of the trie. Therefore during insertion and transformation, the bounding procedure is only applied in the lower half of the trie. Still it is too expensive to do it too often, so we apply the procedure only with a certain probability whenever the insertion or transformation operator accesses a trie node.

3.2 Trie Based on GSR

The trie T_{GSR} is based on the predecessors vector $\Pi = \langle \pi_2, \ldots, \pi_r \rangle$ 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 5 shows an example of how the solution S_1 is stored in T_{GSR}.

Inserting, searching and transforming 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 transformation is $O(r^3)$. 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}.

The bound calculation for T_{GSR} works as follows. Without fixing the full predecessor vector $\Pi = \langle \pi_2, \ldots, \pi_r \rangle$, the global structure in general represents a forest $F^{\text{g}} = \bigcup_{i=1,\ldots,l} K_i^{\text{g}}$ where $K_i^{\text{g}} = \langle V_i^{\text{g}}, T_i^{\text{g}} \rangle$, $i = 1, \ldots, l$, are the l global tree components in F^{g} and V_i^{g} denotes the set of clusters that are contained in component K_i^{g}. For every K_i^{g}, $i = 1, \ldots, l$, that is not a single cluster we apply the dynamic programming procedure that is also used for decoding genotypes in GSR. As a result we obtain for each K_i^{g} the actual nodes to be spanned for the clusters in V_i^{g} that minimize the connection costs. After all global tree components are decoded, they are connected by using the same MST heuristic as in case of SNR. For this purpose the cheapest connections between all components are considered and the condition to connect exactly one node in each cluster is relaxed again. Since each component K_i^{g} is connected in the cheapest way, we obtain a lower bound.

Figure 6 shows an example for this procedure. Assume we want to compute the lower bound for all solutions starting with $\langle 5, 1, 5, \ldots \rangle$. Forest F^{g} contains two global tree components: K_1^{g} with clusters V_1 and V_3 as well as K_2^{g} with clusters V_2, V_4, and V_5. In the next step they are decoded via dynamic programming and connected using the cheapest connections. Assuming we obtain a lower bound of 14 and a previously known solution (S_1) with objective value of 12 exists, the subtrie starting at this point is pruned.

The complexity of calculating lower bounds in GSR is higher than for doing so in SNR. The cheapest connections

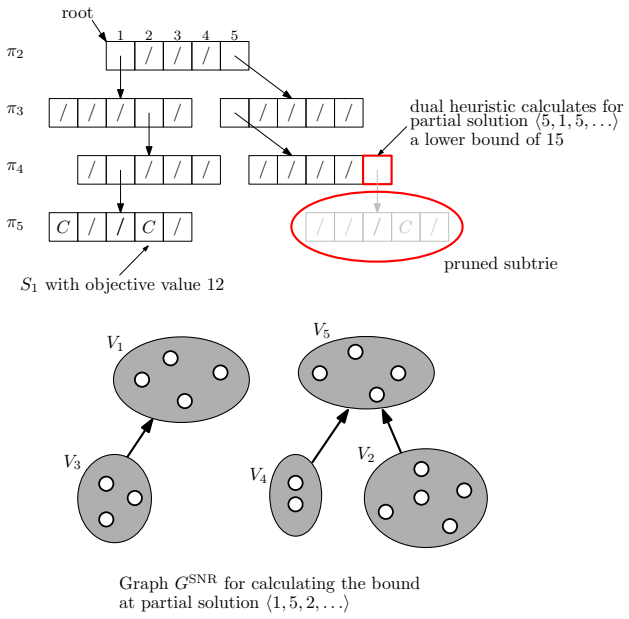


Figure 6: Example of how a lower bound is obtained for a partial solution in GSR and how the corresponding subtrie is pruned.

between clusters and/or nodes are the same which are precomputed for SNR. However, additional effort is necessary for decoding the l global tree components. In overall this requires at most $O(|V|^2)$ time since it will not take longer than decoding a complete genotype in GSR. In preliminary experiments we tried not to decode the components exactly but to use the estimated cheapest connections between clusters instead. Although this was much faster, the obtained bounds were too inaccurate and pruning happened only rarely.

Aside from the higher computational complexity, GSR's lower bounds are typically tighter than those from SNR. On the one hand this is due to the more sophisticated bounding calculation where runtime is not spent in vain. On the other hand decisions in GSR are more powerful per se, since fixing a connection between two widely separated clusters has a larger impact on solution quality than fixing a bad spanned node in a cluster in SNR. While pruning in SNR is typically only possible in the lower half of trie, this does not hold for the lower bounds calculated in GSR. Here we observe that pruning happens on nearly any arbitrary level in the trie, so all levels are considered. However, as in SNR, the bounding calculation is only performed with a certain probability.

4. COMPUTATIONAL RESULTS

We tested our approach on TSPlib[1] instances using geographical center clustering [1]. For each instance we performed 30 independent runs with each considered algorithm variant in order to derive average objective values and standard deviations. When generating new solutions, recombination was always performed and the mutation rate was set to 10%. The population contained 100 solutions. These parameters are relatively standard for an EA. We tested different settings, but the impact on the solution qualities were insignificant. This indicates that the solution archive is able to compensate changes in the primary parameters of the EA and makes it more robust. Archive specific parameters such as the probability for bounding calculations have larger effects. The time and memory overhead can become very large when they are set too high. The appropriate values are derived from extensive testings in [7].

First we show in Table 1 the impact on time and memory consumption with respect to the bounding extension. For this purpose we terminated the EA after a fixed number of 10000 iterations, and we either only used the SNR archive or the GSR archive. This way we are able to see the differences more clearly. The first column lists the instance names with the last three digits indicating $|V|$. For each variant we show average CPU-times required on a single core of an Intel Core 2 quad PC with 2.4GHz and 4GB memory and the average sizes of the archives at the end. Solution qualities are not listed here since they are not the focus of this table. The probability of the bound calculation at a trie node when it is accessed was set to 0.05. Even with this low probability, the time overhead is substantial and in many cases the runtime was doubled. We also observe an increased memory consumption when applying bounding. This might be counterintuitive at first glance since one goal of this technique is to save memory by pruning subtries with inferior solutions. However, a new solution must be generated each time when this happens. So we essentially perform a transformation after pruning which in general introduces new branches in the trie. As pointed out in Section 3.2, the GSR trie is substantially larger than the SNR trie.

Next we extend the results previously published in [9] in order to show the impact of the bounding extension on the solution quality. In Table 2 the EA was tested in the following variants: EA without archive, EA with SNR archive based on trie T_{SNR}, EA with GSR archive based on trie T_{GSR}, EA with full archive, i.e., using both tries, and EA with full archive and bounding. As termination criterion we now used a fixed CPU-time since the overhead caused by the archive and/or the bounding extension must be considered. The first two columns list the instance names 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 variant without archive performs worst in general. Among the two variants where the archive only uses one representation, GSR is more often the better choice. By combining both tries in the archive we are able to get even better results, but including bounding further increases the solution qualities on all except one instance. This indicates that the solution archive and the bounding have a positive impact on the EA, and the time overhead is compensated. Since the differences on $\overline{C(T)}$ for the last two EA variants are relatively small, we performed one-sided Wilcoxon rank sum tests for the assumption that the variant with bounding performs better than the variant without bounding. On the five instances where the results differ, the error probabilities are between 0.07 and 0.50, so we conclude that the differences are not significant.

In order to obtain a more meaningful picture, we compare in Table 3 the last two EA variants on an extended TSPlib instance set[2] introduced in [12]. These instances

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

[2] neumann.hec.ca/chairedistributique/data/gmstp

Table 1: Time and memory consumption by the archive.

	SNR archive				GSR archive			
	no bounding		with bounding		no bounding		with bounding	
Instance	$\overline{\text{time}}$	$\overline{\text{mem}}$	$\overline{\text{time}}$	$\overline{\text{mem}}$	$\overline{\text{time}}$	$\overline{\text{mem}}$	$\overline{\text{time}}$	$\overline{\text{mem}}$
kroa150	23s	6.7 MB	36s	8.0 MB	24s	28.9 MB	37s	32.8 MB
d198	39s	10.7 MB	75s	12.5 MB	41s	55.4 MB	58s	59.5 MB
krob200	49s	8.2 MB	82s	9.7 MB	49s	49.0 MB	99s	54.1 MB
gr202	42s	10.0 MB	68s	11.6 MB	44s	58.3 MB	55s	61.2 MB
ts225	49s	17.3 MB	73s	19.0 MB	53s	84.7 MB	61s	85.3 MB
pr226	64s	6.8 MB	126s	7.7 MB	61s	76.1 MB	100s	89.3 MB
gil262	74s	17.9 MB	123s	21.3 MB	78s	111.4 MB	93s	115.8 MB
pr264	80s	15.1 MB	139s	17.8 MB	83s	108.5 MB	106s	115.9 MB
pr299	101s	18.8 MB	174s	22.1 MB	105s	133.6 MB	129s	137.4 MB
lin318	113s	17.8 MB	217s	22.1 MB	116s	163.5 MB	152s	174.2 MB
rd400	168s	25.8 MB	318s	31.9 MB	178s	264.7 MB	212s	276.9 MB
fl417	204s	18.1 MB	499s	20.5 MB	196s	265.0 MB	326s	302.5 MB
gr431	243s	28.9 MB	466s	34.3 MB	248s	309.3 MB	267s	315.3 MB
pcb442	217s	34.4 MB	406s	41.1 MB	229s	360.0 MB	244s	367.4 MB

Table 2: Results of different EA variants.

		no archive		SNR archive		GSR archive		full archive		with bounding	
Instance	time	$\overline{C(T)}$	sd	$\overline{C(T)}$	sd	$\overline{C(T)}$	sd	$\overline{C(T)}$	sd	$\overline{C(T)}$	sd
kroa150	150s	9830.6	31.4	9831.3	30.1	**9815.0**	0.0	**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	**7044.0**	0.0
krob200	300s	11275.0	45.6	11248.9	7.5	**11244.0**	0.0	**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	**242.0**	0.0
ts225	300s	62290.8	40.4	62299.1	50.9	62268.6	0.5	62268.4	0.5	**62268.3**	0.5
pr226	300s	**55515.0**	0.0	**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	**942.0**	0.0
pr264	450s	21893.2	7.7	21898.4	20.9	**21886.0**	0.0	**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	**20316.0**	0.0
lin318	600s	18545.9	29.2	18547.3	25.6	18525.8	12.4	**18511.0**	10.8	18513.8	9.9
rd400	600s	5953.0	15.4	5959.4	20.2	5946.4	10.8	5940.2	6.5	**5939.7**	6.7
fl417	600s	**7982.0**	0.0	**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	**1033.0**	0.0
pr439	600s	51921.4	60.7	51888.5	56.3	51810.5	26.5	**51791.0**	0.0	**51791.0**	0.0
pcb442	600s	19717.0	59.5	19708.1	70.2	19632.6	21.1	19623.7	15.9	**19617.0**	12.4

Table 4: Comparison with other state-of-the-art approaches on the standard TSPlib instances.

	TS		VNS			DCS			EA + archive		
Instance	C(T)	time	$\overline{C(T)}$	sd	time	$\overline{C(T)}$	sd	time	$\overline{C(T)}$	sd	time
kroa150	**9815.0**	150s	**9815.0**	0.0	150s	**9815.0**	0.0	133s	**9815.0**	0.0	39s
d198	7062.0	300s	**7044.0**	0.0	300s	**7044.0**	0.0	265s	**7044.0**	0.0	78s
krob200	11245.0	300s	**11244.0**	0.0	300s	**11244.0**	0.0	265s	**11244.0**	0.0	78s
gr202	**242.0**	300s	**242.0**	0.0	300s	**242.0**	0.0	265s	**242.0**	0.0	78s
ts225	62366.0	300s	62268.5	0.5	300s	**62268.3**	0.5	265s	62268.5	0.5	78s
pr226	**55515.0**	300s	**55515.0**	0.0	300s	**55515.0**	0.0	265s	**55515.0**	0.0	78s
gil262	**942.0**	450s	942.3	1.0	450s	**942.0**	0.0	398s	**942.0**	0.0	118s
pr264	**21886.0**	450s	21886.5	1.8	450s	**21886.0**	0.0	398s	**21886.0**	0.0	118s
pr299	20339.0	450s	20322.6	14.7	450s	20317.4	1.5	398s	**20316.0**	0.0	118s
lin318	18521.0	600s	**18506.8**	11.6	600s	18513.6	7.8	531s	18519.3	8.4	157s
rd400	5943.0	600s	5943.6	9.7	600s	5941.5	9.9	531s	**5939.5**	5.2	157s
fl417	7990.0	600s	**7982.0**	0.0	600s	7982.7	0.5	531s	**7982.0**	0.0	157s
gr431	1034.0	600s	**1033.0**	0.2	600s	**1033.0**	0.0	531s	**1033.0**	0.0	157s
pr439	51852.0	600s	51847.9	40.9	600s	51833.8	36.1	531s	**51791.0**	0.0	157s
pcb442	19621.0	600s	19702.8	52.1	600s	19662.5	39.8	531s	**19626.0**	17.0	157s

Table 5: Comparison with other state-of-the-art approaches on the extended TSPlib instances.

	GA		TS2		EA + archive			
Instance	$C^*(T)$	time	$C^*(T)$	time	$C^*(T)$	$\overline{C(T)}$	sd	time
ali535	114379	492s	**114303**	683s	**114303**	114419.1	96.6	243s
att532	12007	500s	**12001**	597s	**12001**	12007.8	6.3	115s
d493	16526	388s	**16493**	587s	**16493**	16501.0	18.3	154s
d657	19465	969s	**19427**	1056s	**19427**	19456.6	32.5	335s
fl417	7936	218s	**7935**	233s	**7935**	7935.0	0.0	2570s
gil262	**887**	73s	**887**	74s	**887**	887.0	0.0	13s
gr431	86899	266s	**86885**	233s	**86885**	86903.4	42.3	80s
gr666	144918	2866s	144756	1365s	**144737**	144747.7	40.8	237s
lin318	18476	105s	**18471**	130s	**18471**	18486.3	4.3	46s
p654	22214	881s	22208	1045s	**22207**	22207.0	0.0	1634s
pa561	868	559s	**864**	702s	865	870.7	2.7	107s
pcb442	19670	284s	**19571**	266s	**19571**	19593.9	22.1	66s
pr264	21886	57s	**21872**	72s	**21872**	21872.0	0.0	20s
pr299	20307	86s	**20290**	94s	**20290**	20290.0	0.0	21s
pr439	51808	981s	51760	574s	**51749**	51749.9	3.6	84s
rat575	2189	627s	**2170**	762s	**2170**	2180.0	6.3	125s
rat783	3044	1653s	3017	1916s	**3015**	3027.4	7.4	292s
rd400	5880	205s	**5868**	208s	**5868**	5875.9	8.9	56s
si535	**12791**	458s	**12791**	573s	**12791**	12791.0	0.0	123s
u574	15069	620s	15037	517s	**15034**	15058.1	14.3	155s
u724	16015	1281s	15905	1290s	**15904**	15947.4	26.6	324s

Table 3: Results of different EA variants on the extended TSPlib instances.

		no bounding		with bounding		sign.diff.
Instance	time	$\overline{C(T)}$	sd	$\overline{C(T)}$	sd	error
ali535	600s	114581.1	95.8	**114404.9**	90.7	<0.001
att532	600s	12008.1	7.4	**12004.9**	4.5	0.051
d493	600s	16516,8	14,5	**16494.9**	3.1	<0.001
d657	600s	19504.0	42.2	**19451.7**	31.3	<0.001
fl417	600s	**7935.0**	0.0	**7935.0**	0.0	N.A.
gil262	450s	**887.0**	0.0	**887.0**	0.0	N.A.
gr431	600s	86889.2	18.0	**86888.2**	17.3	0.500
gr666	600s	144837.3	109.1	**144790.6**	77.1	0.037
lin318	600s	18485.9	13.9	**18485.1**	8.6	0.468
p654	600s	**22207.0**	0.0	**22207.0**	0.0	N.A.
pa561	600s	870.6	2.9	**866.7**	2.4	<0.001
pcb442	600s	19589.0	21.5	**19584.1**	15.0	0.109
pr264	450s	**21872.0**	0.0	**21872.0**	0.0	N.A.
pr299	450s	20301.2	16.3	**20290.0**	0.0	<0.001
pr439	600s	51754.3	15.1	**51749.0**	0.0	0.050
rat575	600s	2184.0	6.3	**2178.5**	4.8	<0.001
rat783	600s	3033.6	13.2	**3028.5**	8.3	0.076
rd400	600s	**5874.9**	11.2	5875.0	5.4	0.670
si535	600s	**12791.0**	0.0	**12791.0**	0.0	N.A.
u574	600s	15063.0	15.1	**15051.4**	8.9	<0.001
u724	600s	15965.1	35.4	**15949.2**	29.4	0.046

are larger and have also been derived by geographical center clustering. Note that some of these instances have the same names as those in the first set, but the clustering data and/or the distance data are different. Beside the average final solution values and corresponding standard deviations, we also list for each instance the error probability (error) of the one-sided Wilcoxon rank sum test for the assumption that the variant with bounding performs better than the variant without bounding. Besides five instances where both variants perform equally well and one instance where the EA variant without bounding performs slightly better, we observe that the bounding extension now consistently increases the solution quality. On nine instances these improvements are statistically significant with error probabilities less than 5%.

In Table 4 we compare our EA using the full archive and bounding with several leading state-of-the-art approaches from literature consisting of a tabu search approach (TS1) by Ghosh [5], a hybrid variable neighborhood search approach (VNS) by Hu et al. [8], and an algorithm based on dynamic candidates sets (DCS) by Jiang and Chen [10]. Using a fixed CPU-time as termination criterion, TS1 and VNS ran on a Pentium 4 PC with 2.8GHz, DCS ran on a Pentium D PC with 2.66GHz and the EA ran on an Intel Core i7 PC with 3.4GHz (in contrast to the preliminary tests where an Intel Core 2 quad PC was used). Since the different approaches used different hardware, we scaled the time limits according to the the well-known Standard Performance Evaluation Corporation (SPEC) benchmark[3]. It indicates that Core i7 (score 47.1) is around 332% faster than Pentium D (score 14.2) which is then again around 15% faster than Pentium 4 (score 12.3). After compensating the heterogeneous testing environments by using the appropriate time limits, we observe that our EA can compete well with the other approaches, especially on larger instances.

In Table 5 we compare our approach with a genetic algorithm (GA) by Golden et al. [6] and a tabu search approach (TS2) by Oencan et al. [12] on the extended TSPlib instances. GA ran on a Xeon workstation with 2.66GHz and TS2 ran on a Pentium 4 PC with 3GHz. Since these authors did not use a fixed CPU-time as termination criterion, we terminated our EA after 2000 iterations without improvement. Because only best solutions are listed for GA and

[3]www.spec.org/cpu2006

TS2 in [12], we also focused on the best solutions obtained by 30 runs of our EA (but we nevertheless added the average values and the standard deviations). We observe that our approach competes well with TS2 and outperforms the GA. By means of the SPEC benchmarks for the different hardwares on which TS2 and our EA were tested, we can see that the required CPU-times are comparable for most of the instances. On the instance fl417 our EA consumes an exceptionally large amount of runtime. The reason is that the final best solution can be found extremely easily, therefore the solution archive spends much time on converting all the duplicates which arise.

5. CONCLUSIONS AND FUTURE WORK

In this paper we proposed a novel bounding extension for a solution archive enhanced EA for the GMSTP. Since the EA uses operators based on two dual representations, we introduced two different bounding heuristics, respectively. Tests on TSPlib instances show that the extension, particularly when combining both representations, is able to improve the search behavior significantly. Compared with several leading state-of-the-art approaches from literature, the archive-enhanced EA is able to keep up with the very top candidates.

For future work, we want to investigate another extension for solution archives where the transformation of duplicate solutions is guided by estimation heuristics. This approach complements the bounding extensions so that the solution archive not only cuts off inferior solutions, but puts more focus on promising ones. We also believe that the concept of solution archives is a powerful addition to EAs in a more general sense, when it is implemented adequately for appropriate combinatorial optimization problems. Hence we want to further pursue this concept also for other problems.

Acknowledgements

We thank Markus Wolf, Mika Sonnleitner, and Christian Gruber, who helped in the implementation of the described concepts and did the testing as part of their master theses [17, 16, 7].

6. REFERENCES

[1] C. Feremans. *Generalized Spanning Trees and Extensions*. PhD thesis, Universite Libre de Bruxelles, Belgium, 2001.

[2] C. Feremans, M. Labbe, and G. Laporte. A comparative analysis of several formulations for the generalized minimum spanning tree problem. *Networks*, 39(1):29–34, 2002.

[3] C. Feremans, M. Labbe, and G. Laporte. The generalized minimum spanning tree problem: Polyhedral analysis and branch-and-cut algorithm. *Networks*, 43(2):71–86, 2004.

[4] E. Fredkin. Trie memory. *Communications of the ACM*, 3:490–499, 1960.

[5] 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.

[6] 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.

[7] C. Gruber. Ein Lösungsarchiv mit Branch-and-Bound-Erweiterung für das Generalized Minimum Spanning Tree Problem. Master's thesis, Vienna University of Technology, 2011.

[8] 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.

[9] B. Hu and G. R. Raidl. An evolutionary algorithm with solution archive for the generalized minimum spanning tree problem. In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, *Proceedings of the 13th International Conference on Computer Aided Systems Theory: Part I*, volume 6927 of *LNCS*, pages 287–294. Springer, 2012.

[10] 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.

[11] Y. S. Myung, C. H. Lee, and D. W. Tcha. On the generalized minimum spanning tree problem. *Networks*, 26:231–241, 1995.

[12] T. Öncan, J.-F. Cordeau, and G. Laporte. A tabu search heuristic for the generalized minimum spanning tree problem. *European Journal of Operational Research*, 191(2):306–319, 2008.

[13] P. C. Pop. *The Generalized Minimum Spanning Tree Problem*. PhD thesis, University of Twente, The Netherlands, 2002.

[14] 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.

[15] G. R. Raidl and B. A. Julstrom. Edge-sets: An effective evolutionary coding of spanning trees. *IEEE Transactions on Evolutionary Computation*, 7(3), 2003.

[16] M. Sonnleitner. Ein neues Lösungsarchiv für das Generalized Minimum Spanning Tree-Problem. Master's thesis, Vienna University of Technology, 2010.

[17] M. Wolf. Ein Lösungsarchiv-unterstützter evolutionärer Algorithmus für das Generalized Minimum Spanning Tree-Problem. Master's thesis, Vienna University of Technology, 2009.

[18] 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.