Combining a Memetic Algorithm with Integer Programming to Solve the Prize-Collecting Steiner Tree Problem*

Gunnar W. Klau[1], Ivana Ljubić[1], Andreas Moser[1], Petra Mutzel[1], Philipp Neuner[1], Ulrich Pferschy[2], Günther Raidl[1], and René Weiskircher[1]

[1] Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstraße 9–11/186, 1040 Vienna, Austria
{klau,ljubic,moser,mutzel,neuner,raidl,weiskircher}@ads.tuwien.ac.at
[2] Department of Statistics and Operations Research University of Graz, Austria
pferschy@uni-graz.at

Abstract. The prize-collecting Steiner tree problem on a graph with edge costs and vertex profits asks for a subtree minimizing the sum of the total cost of all edges in the subtree plus the total profit of all vertices **not** contained in the subtree. For this well-known problem we develop a new algorithmic framework consisting of three main parts:
(1) An extensive preprocessing phase reduces the given graph without changing the structure of the optimal solution. (2) The central part of our approach is a memetic algorithm (MA) based on a steady-state evolutionary algorithm and an exact subroutine for the problem on trees. (3) The solution population of the memetic algorithm provides an excellent starting point for post-optimization by solving a relaxation of an integer linear programming (ILP) model constructed from a model for finding the minimum Steiner arborescence in a directed graph.
Extensive experiments on benchmark instances from the literature show that our combination of an MA with ILP-based post-optimization compares favorably with previously published results. While our solution values are almost always the same (not surprisingly, since an extension of our ILP approach shows the optimality of these values), we obtain a significant reduction of running time for medium and large instances.

1 Introduction

We consider the prize-collecting Steiner tree problem, an extension of the well-known *Steiner problem*, where the input is a graph whose vertices are associated with profits and edges with costs. Our goal is to find a connected subgraph that minimizes the sum of the profits of the vertices that are **not** contained in the subgraph plus the costs of the edges in the subgraph. The problem finds

* Partly supported by the Doctoral Scholarship Program of the Austrian Academy of Sciences (DOC) and by the Austrian Science Fund (FWF), grant P16263-N04.

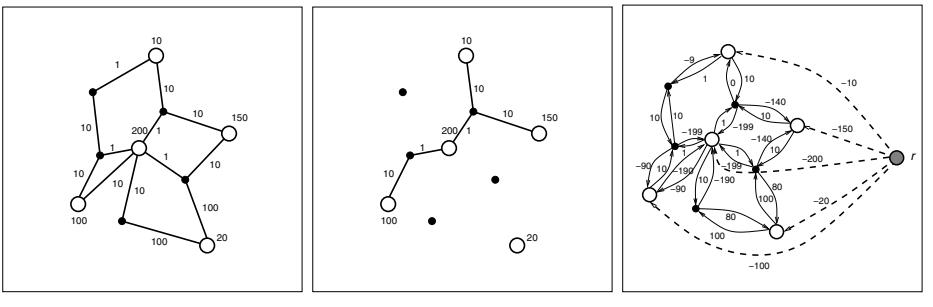
K. Deb et al. (Eds.): GECCO 2004, LNCS 3102, pp. 1304–1315, 2004.

its application in the design of networks for communication or distribution of utilities such as district heating or water.

Let $G = (V, E, c, p)$ be an undirected connected graph with $p : V \to \mathbb{R}^{\geq 0}$ a profit function on the vertices and $c : E \to \mathbb{R}^{\geq 0}$ a cost function on the edges. The *prize-collecting Steiner tree problem* (PCSTP) is to find a connected subgraph $T = (V_T, E_T)$ of G, that minimizes

$$c(T) = \sum_{v \notin V_T} p_v + \sum_{e \in E_T} c_e. \tag{1}$$

Note that if the goal is to find a subgraph T that **maximizes** the sum of the profits of the vertices in T minus the cost of the edges in T, every optimal solution is an optimal solution for our minimization problem and vice versa. Furthermore, it is easy to see that every optimal solution T is a tree. Throughout this paper we will distinguish between *positive vertices*, defined as $R = \{v \in V \mid p_v > 0\}$, and *non-positive vertices*. An example of a PCSTP instance and its feasible solution are shown in Figure 1(a) and 1(b), respectively.



(a) (b) (c)

Fig. 1. Example of a PCSTP instance. Each connection has fixed costs, hollow circles and filled circles represent positive and non-positive vertices, respectively (Fig. 1(a)). Figure 1(b) shows a feasible solution and Figure 1(c) the transformation into the Steiner arborescence problem.

Previous Work. The PCSTP has been introduced by Bienstock et al. [1], where a factor 3 approximation algorithm has been proposed. Several other approximation algorithms have been developed (see [7,8]). Segev [16] defined the *node weighted Steiner tree problem* (NWSTP) – another extension of the Steiner problem in graphs, where, in contrast to PCSTP, some vertices must be contained in every solution. Polyhedral studies of this problem can be found in [5,6]. Engevall et al. [4] proposed a Lagrangean relaxation approach based on the *shortest spanning tree* integer linear programming (ILP) formulation for NWSTP.

Lucena and Resende [11] presented a cutting plane algorithm for solving PCSTP based on generalized subtour elimination constraints. The algorithm

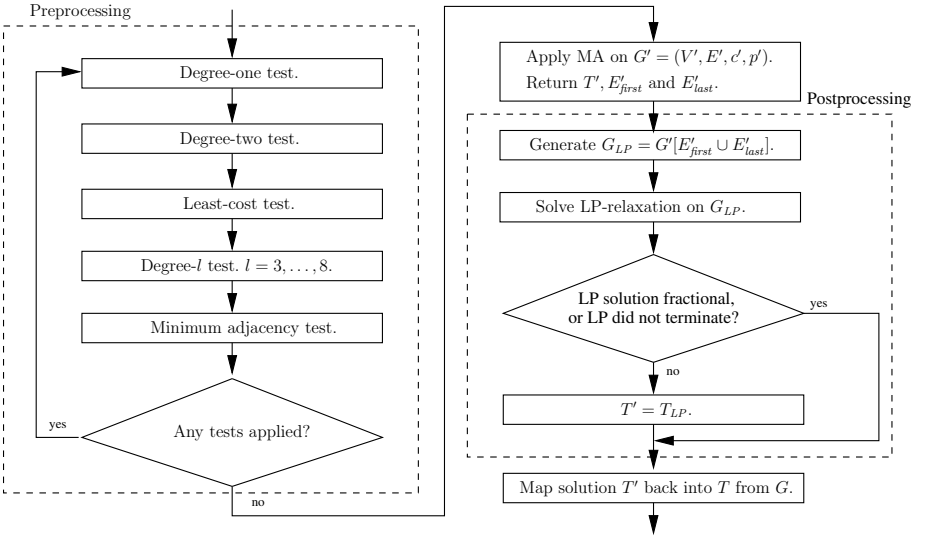


Fig. 2. Three main phases of the proposed approach for PCSTP: (1) Preprocessing reduces the given input graph $G = (V, E, c, p)$ into $G' = (V', E', c', p')$ without changing the structure of the optimal solution. (2) A memetic algorithm (MA). (3) A collection of solutions of the MA provides an excellent starting point for post-optimization by solving a relaxation of an ILP model constructed from a model for finding the minimum Steiner arborescence in a directed graph.

also contains basic reduction steps similar to those already proposed by Duin and Volgenant [3] for NWSTP.

Canuto et al. [2] developed a multi-start local-search-based algorithm with perturbations for PCSTP. It comprises Goemans-Williamson's algorithm, 1-flip neighborhood search and path relinking. A variable neighborhood search method is applied as a post-optimization procedure. The algorithm found optimal solutions on nearly all instances from [11] for which the optima were known.

Our Contribution. A new algorithmic framework is developed as outlined in Figure 2. The computational results given in Section 3 show that our new approach is significantly faster than the previous approach by Canuto et al. [2] while the solutions have the same quality. For a number of instances we manage to find new best solutions, while on the majority of instances our solution values are identical, which is not surprising: Extending our ILP approach shows that these values are indeed optimal. The progress we obtain with respect to running time gives rise to the possibility of solving much larger instances in the future.

2 Combining the Memetic Algorithm with an ILP Model

Within this section, we propose basic ideas of our new algorithmic framework for the PCSTP whose outline is given in Fig. 2. After the input graph G has

been reduced into a graph $G' = (V', E', c', p')$, we apply a memetic algorithm that uses problem-dependent operators and strongly interacts with an exact subroutine for the PCSTP problem on trees.

Our ILP-based post-optimization procedure utilizes the combined context of the MA-solutions to produce a final tree that is superior to any single one in the population. Furthermore, the post-optimization algorithm benefits from the fact that solving the PCSTP restricted to a sparse edge set can be much simpler than solving the original problem.

As input for the ILP algorithm, we take a subgraph G_{LP} of G' induced by $E_{LP} = E'_{first} \cup E'_{last}$, the sets of edges that appear in any single solution of the first, respectively, last population. Note that taking the edges from the first generation enables us to escape local optima found by MA.

The best-found subtree T of the original graph G is finally determined by mapping back the solution T' found by the ILP-relaxation.

2.1 Preprocessing

In this section, we briefly describe reduction techniques adopted from the work of Duin and Volgenant [3] for the NWSTP, which have been partially used also in [11]. From the implementation point of view, we transform the graph $G = (V, E, c, p)$ into a reduced graph $G' = (V', E', c', p')$ by applying the steps described below and maintain a *backmapping* function to transform each feasible solution T' of G' into a feasible solution T of G.

Least-Cost Test. Let d_{ij} represent the shortest path length between any two vertices i and j from V (considering only edge-costs). If $\exists e = (i, j)$ such that $d_{ij} < c_{ij}$ then edge e can simply be discarded from G. The procedure's time complexity is dominated by the computation of all-pair shortest paths, which is $O(|E||V| + |V|^2 \log |V|)$ in the worst case.

Degree-l Test. Consider a vertex $v \notin R$ of degree $l \geq 3$, connected to vertices from $Adj(v) = \{v_1, v_2, \ldots, v_l\}$. For any subset $K \subset V$, denote with $\mathrm{MST}_d(K)$, the minimum spanning tree of K with distances d_{ij}. If

$$\mathrm{MST}_d(K) \leq \sum_{w \in K} c_{vw}, \quad \forall K \subseteq Adj(v), \quad |K| \geq 3, \tag{2}$$

then v's degree in an optimal solution must be zero or two. Hence, we can remove v from G by replacing each pair (v_i, v), (v, v_j) with (v_i, v_j) either by adding a new edge $e = (v_i, v_j)$ of cost $c_e = c_{v_i v} + c_{v v_j} - p_v$ or in case e already exists, by defining $c_e = \min\{c_e, c_{v_i v} + c_{v v_j} - p_v\}$.

The procedure's worst case running time is dominated by the computation of all-pair shortest paths, which is $O(|E||V| + |V|^2 \log |V|)$. It is straightforward to apply a simplified version of this test to all vertices $v \in V$ with $l = 1$ and $l = 2$.

Minimum Adjacency Test. This test is also known as $V \setminus K$ *reduction test* from [3]. If there are adjacent vertices $i, j \in R$ such that:

$$\min\{p_i, p_j\} - c_{ij} > 0 \text{ and } c_{ij} = \min_{it \in E} c_{it},$$

then i and j can be fused into one vertex of weight $p_i + p_j - c_{ij}$.

Summary of the Preprocessing Procedure. We apply the steps described above iteratively, as long as any of them changes the input graph (see Fig. 2). The total number of iterations is bounded by the number of edges in G. Each iteration is dominated by the time complexity of the least-cost test. Thus, the preprocessing procedure requires $O(|E|^2|V| + |E||V|^2 \log |V|)$ time in the worst case, in which the input graph would be reduced to a single vertex. However, in practice, the running time is much lower, as documented in Section 3. The space complexity of preprocessing does not exceed $O(|E|^2)$.

2.2 A Memetic Algorithm for the PCSTP

For many hard combinatorial optimization problems, combinations of evolutionary algorithms and problem-dependent heuristics, approximation algorithms or local improvement techniques have been applied with great success. In a memetic algorithm (MA), candidate solutions created by an evolutionary algorithm framework are fine-tuned by some of these procedures [13].

We propose an MA based on a straight-forward steady-state evolutionary algorithm combined with an exact algorithm for solving the PCSTP on trees. In each iteration, we apply k-ary tournament selection with replacement in order to select two parental solutions for mating. A new candidate solution is always created by recombining these parents, mutating it with probability $p_{\text{mut}} \in [0, 1]$, and pruning the obtained tree to optimality. Such a solution replaces always the worst solution in the population with one exception: To guarantee a minimum diversity, a new candidate whose set of edges $E_{T'}$ is identical to that of a solution already contained in the population is discarded [14].

Each randomly created initial solution and each solution derived by recombination and possibly mutation is optimally pruned with respect to its subtrees, using the local improvement algorithm described below.

Local Improvement. The algorithm we use here solves tree instances of the PCSTP to optimality and runs in $O(|V'|)$ time (see also [8,10]).

Given a tree instance $T' = (V_{T'}, E_{T'}, p', c')$ created by an MA, a subtree of T' is *optimal*, if there is no subtree of T' with costs lower than $c(T')$. The algorithm we use here maximizes the sum of the profits of the vertices in T' minus the sum of the edge-costs in T'. We label the vertices $v \in V_{T'}$ and traverse them in bottom-up order, until we end-up with a single vertex. Finally, the optimal solution corresponds to the subtree shrunk within the vertex v^* such that $v^* = \arg\max_{v \in V_{T'}} l_v$. The algorithm is as follows:

1. Set $l_v = p'_v$, for all $v \in V_{T'}$;
2. For all leaves $u \in V_{T'}$: (a) if $c'_{uv} \leq l_u$, shrink u and v into one vertex and set $l_v = l_v + l_u - c'_{uv}$; (b) Delete u;
3. Goto 2. until a single vertex is left;

Clustering. Employing *clustering* as a grouping procedure within variation operators, we can group the subsets of vertices and insert or delete them at once. For each positive vertex $z \in R'$, we define a cluster set $N(z)$ [12]:

$$N(z) := \{v \in V' \setminus R' \mid \forall c \in R' : \ d'_{vz} \leq d'_{vc}\} \cup \{z\},$$

where d'_{vz} denotes the shortest path length between v and z. Hence, each non-positive vertex v is assigned to the cluster set of its nearest positive vertex $z = \mathit{base}(v)$. Note that the sets $N(z)$ are analogous to Voronoi regions in the Euclidean plane.

Mehlhorn [12] proposed an efficient implementation of the clustering algorithm which runs in $O(|V'| \log |V'| + |E'|)$ time.

Edge-Set Encoding. From spanning tree problems, we know that a direct representation of spanning trees as sets of their edges exhibits significant advantages over indirect encodings [15]. In our approach, the PCSTP solution edges are stored in hash-tables, requiring only $O(|V'|)$ space. Thus, insertion and deletion of edges, as well as checking for existence of an edge, can be done in expected constant time.

Initialization. Given an input graph $G' = (V', E', c', p')$ and its set of positive vertices R', the *distance network* $G_D(R', E_D, c_D)$ is an undirected complete graph whose edge costs $c_D(u, v)$ are given by the shortest path lengths between u and v in G'. For generating initial solutions we use the following modification of the *distance network heuristic* for the Steiner tree problem [12]:

1. Randomly select a subset $V'_{init} \subset R'$ of size $\lceil p_{init} \cdot |R'| \rceil, p_{init} \in (0, 1)$;
2. Construct the minimum spanning tree (MST) T'_{init} on the subgraph of G_D induced by V'_{init};
3. Replace each edge of T'_{init} by its corresponding shortest path in G' to obtain $G'_r = (V'_r, E'_r)$;
4. Find the MST T'_r on the subgraph of G' induced by V'_r;
5. Apply the exact algorithm for trees to solve T'_r to optimality;

Recombination. The recombination operator is designed with strong inheritance in mind; we try to adopt the structural properties of two parental solutions. If the two solutions to be combined share at least one vertex, we just construct the spanning tree over the union of their edge sets. Due to the deterministic nature of our local improvement subroutine, we build a random spanning tree on the union of parental edges to avoid premature convergence.

When the parent solutions are disjoint, we randomly choose a vertex out of each solution, look up the shortest path between these two vertices and add for each vertex v along the path all the edges that belong to cluster $N(\mathit{base}(v))$. Finally, we build a random spanning tree over all these edges and apply local improvement.

Mutation. The aim of the mutation operator is to make small changes in the current solution which we achieve by connecting one cluster to the solution. To find an appropriate cluster to add, the algorithm randomly chooses a *border vertex* v which is a vertex adjacent to at least one vertex outside our current solution. We incorporate the vertices of cluster $N(base(v))$ into our solution and search for a neighboring cluster whose base vertex v' is preferably not yet an element of the current solution; the vertices of $N(base(v'))$ will be added to our solution. Finally we construct a minimum spanning tree and apply local improvement.

Assuming the complete distance network is determined once in the preprocessing phase and its edges are pre-sorted in non-increasing order, as well as the edges of E', the running time complexity of initialization and variation operators is $O(|E'| \cdot \alpha(|E'|, |V'|))$.

2.3 ILP Formulation

Our ILP formulation relies on a transformation of the PCSTP to the problem of finding a minimum subgraph in a related, directed graph as proposed by Fischetti [5]. We transform the graph $G_{\text{ILP}} = (V_{\text{ILP}}, E_{\text{ILP}}, c', p')$ that results from the application of the memetic algorithm as described in Section 2.2 into the directed graph $G'_{\text{ILP}} = (V_{\text{ILP}} \cup \{r\}, A_{\text{ILP}}, c'')$ (see Figure 1(c) for an example).

In addition to the vertices of the input graph G_{ILP}, the vertex set of the transformed graph contains an artificial root r. The arc set A_{ILP} contains two directed edges (v, w) and (w, v) for each edge $(v, w) \in E_{\text{ILP}}$ plus a set of arcs from the root r to the positive vertices $\{v \in V_{\text{ILP}} \mid p_v > 0\}$. We define the cost vector c'' as follows:

$$c''_{vw} = c'_{vw} - p'_w \quad \forall (v, w) \in A_{\text{ILP}}, v \neq r \quad \text{and} \quad c''_{rv} = -p'_v \quad \forall (r, v) \in A_{\text{ILP}} \ .$$

A subgraph T_{ILP} of G'_{ILP} that forms a directed tree rooted at r is called a *Steiner arborescence*. It is easy to see that such a subgraph corresponds to a solution of the PCSTP if r has degree 1 in G'_{ILP} (*feasible arborescence*). In particular, a feasible arborescence with minimal total edge cost corresponds to an optimal prize-collecting Steiner tree.

We model the problem of finding a minimum Steiner arborescence T_{ILP} by means of an integer linear program. Therefore, we introduce a variable vector $x \in \{0, 1\}^{|A_{\text{ILP}}|+|V_{\text{ILP}}|}$ with the following interpretation:

$$x_{vw} = \begin{cases} 1 & (v, w) \in T_{\text{ILP}} \\ 0 & \text{otherwise} \end{cases} \forall (v, w) \in A_{\text{ILP}}, \quad x_{vv} = \begin{cases} 1 & v \notin T_{\text{ILP}} \\ 0 & \text{otherwise} \end{cases} \quad \forall v \in V_{\text{ILP}} \setminus \{r\}$$

The ILP is then as follows:

$$\min \quad \sum_{a \in A_{\mathrm{ILP}}} c''_a x_a \tag{3}$$

$$\text{subject to} \quad x(\delta^-(\{v\})) + x_{vv} = 1 \qquad \forall v \in V_{\mathrm{ILP}} \setminus \{r\} \tag{4}$$

$$x(\delta^-(S)) \geq 1 - x_{vv} \qquad v \in S, r \notin S, \forall S \subset V_{\mathrm{ILP}} \tag{5}$$

$$\sum_{(r,v) \in A_{\mathrm{ILP}}} x_{rv} \leq 1 \tag{6}$$

$$x_{vw}, x_{vv} \in \{0,1\} \qquad \forall (v,w) \in A_{\mathrm{ILP}}, \forall v \in V_{\mathrm{ILP}}, \tag{7}$$

where $\delta^-(S) = \{(u,v) \in A_{\mathrm{ILP}} \mid u \notin S, v \in S\}$.

Constraint (4) states that every vertex that is part of the solution must have at least one incoming edge while (5) states that for each vertex v in the solution, there must be a directed path from r to v. Constraint (6) ensures that at most one of the edges starting at the artificial root is chosen. We use CPLEX as linear program solver to solve the *ILP-relaxation* of the problem obtained by replacing constraints (7) with $0 \leq x_{vw}, x_{vv} \leq 1$, $(v,w) \in A_{\mathrm{ILP}}, v \in V_{\mathrm{ILP}}$.

There are exponentially many constraints of type (5), so we do not insert them at the beginning but rather *separate* them during the optimization process; that is, we only add constraints violated by the current solution of the ILP-relaxation. These violated constraints can be found efficiently using a maximum flow algorithm on the graph with arc-capacities given by the current solution. We also use *pricing* which means that we do not start with all the variables but rather add them only if needed to prove optimality. A detailed description of this approach that also includes *flow-balance* and *asymmetry constraints* can be found in [9].

3 Computational Results

We tested our new approach extensively on 114 benchmark instances[1] described in [2,11]. The instances range in size from 100 vertices and 284 edges to 1000 vertices and 25 000 edges. Because of space limitations, we present detailed results for the 60 most challenging instances from Steiner series `C` and `D`. Graphs from series `C` have 500, and graphs from series `D` 1000 vertices. Table 1 lists the instance name, its number of edges $|E|$, the size of the graph after the reductions described in Section 2.1 ($|V'|$, $|E'|$) and the time spent on preprocessing (t_p [s]).

The following setup was used for the memetic algorithm as it proved to be robust in preliminary tests: Population size $|P| = 800$; group size for tournament selection $k = 5$; parameter for initializing solutions $p_{init} = 0.9$; mutation probability $p_{\mathrm{mut}} = 0.3$. Each run was terminated when no new best solution could be identified during the last $\Omega = 10\,000$ iterations.

Because of its stochastic nature, the MA was performed 30 times on each instance and the average results are presented in Table 1 which also contains the

[1] Benchmark instances are available from `http://research.att.com/~mgcr/data/`.

average costs $c(T)_{avg}$ and their standard deviation $\sigma(c)$. Furthermore, we show the average CPU-time and the average number of evaluated solutions until the best solution was found (t, respectively *evals*), and the success rates (sr [%]), i.e. the percentage of instances for which optimal solutions could be found.

We also list the results of our combined approach, MA+ILP, where one MA run (with a fixed seed-value) was post-optimized with the ILP method. The value of the obtained solution and *only* the post-optimization CPU-time in seconds are given in columns $c(T)$ and t [s], respectively. Note that the time presented for MA excludes preprocessing times.

We compared the results of our new approach (MA+ILP) to those of Canuto et al. (CRR) obtained using multi-start local search with perturbations and variable neighborhood search [2]. Table 1 provides the solution values of CRR ($c(T)$) and the total running time in seconds (t). In most cases our solution values are identical to CRR. The cases where one of the two is superior are marked by a box.

Finally, to see if we can obtain provably optimal solutions using the ILP approach, we continued the optimization: starting from the ILP-solution of the restricted MA+ILP problem, the rest of variables from G' was considered within pricing of the ILP-relaxation. In column OPT, we show the values of the obtained integer solutions. If we did not obtain an integer solution, or if our ILP-based algorithm terminated abnormally (because of memory consumption) we show the values obtained by Lucena & Resende [11], denoting it with $^+$, respectively *. Note that all values given in OPT are optimal except for `D14-B` where the best-known lower bound is printed [11]. The last column t [s] lists the *additional* CPU-time needed to compute a provably optimal solution.

When comparing our running time data (achieved on a Pentium IV with 2.8 GHz, 2 GB RAM, SPECint2000=1204) with the results of Canuto et al. [2] (Pentium II with 400 MHz, 64 MB RAM), the widely used SPEC© performance evaluation (`www.spec.org`) does not provide a direct scaling factor. However, taking a comparison to the respective benchmark machines both for SPEC 95 and SPEC 2000 into account, we can argue by a conservative estimate that dividing the Canuto et al. running times by a factor of 10 gives a very reasonable basis of comparison to our data.

Table 2 summarizes our results over all benchmark instances used in [2]. The second and third column show that using sophisticated preprocessing reduces the number of nodes and edges in the problem graph by 30-45% on average. We also provide the average quality (%-*gap*) and the average *total* running time for the approach of Canuto et al. (CRR), our memetic algorithm (MA) and the MA combined with linear programming post-processing (MA+ILP), respectively. The last column gives the average running time for computing a provably optimal solution with our ILP-based approach or a question mark where we could not find an optimal solution for all instances.

The summarized results indicate that MA alone is substantially faster than CRR (by an order of magnitude for the largest group `D`), but the average solution quality is slightly worse. Solutions of MA+ILP are not significantly worse than CRR solutions, but MA+ILP is much faster than CRR, even when we take the difference in hardware into account.

Table 1. Results obtained by Canuto et al. (CRR), the memetic algorithm (MA) and the combination of MA with ILP (MA+ILP) on selected instances from Steiner series C and D. Running times in (CRR) to be divided by 10 for comparison (cf. SPEC comparison).

	Orig.	Preprocessing			MA					MA+ILP		CRR		OPT-ILP	
Instance	$\|E\|$	$\|V'\|$	$\|E'\|$	t_p [s]	$c(T)_{avg}$	$\sigma(c)$	t [s]	*evals*	*sr* [%]	$c(T)$	t [s]	$c(T)$	t [s]	*OPT*	t [s]
C11-A	2500	489	2143	9.4	18.0	0.0	6.1	500	100.0	18	0.4	18	128	18	0.2
C11-B	2500	489	2143	9.5	32.0	0.0	9.1	1103	100.0	32	0.4	32	140	32	4.7
C12-A	2500	484	2186	6.8	38.7	0.5	9.0	2456	33.3	38	0.4	38	162	38	0.3
C12-B	2500	484	2186	6.8	46.0	0.0	8.7	590	100.0	46	0.5	46	156	46	0.8
C13-A	2500	472	2113	9.8	237.0	0.2	17.9	5326	0.0	**236**	0.6	237	1050	236	0.5
C13-B	2500	471	2112	9.8	258.5	0.7	35.9	15455	60.0	258	18.5	258	733	258	52.5
C14-A	2500	466	2081	7.5	293.0	0.0	21.0	3163	100.0	293	1.7	293	829	293	0.4
C14-B	2500	459	2048	7.5	318.6	0.5	29.8	9211	43.3	318	1.0	318	766	318	0.4
C15-A	2500	406	1871	6.5	502.2	0.8	45.4	14727	20.0	501	4.7	501	957	501	0.5
C15-B	2500	370	1753	6.0	551.8	0.9	45.7	15607	46.7	551	0.8	551	837	551	0.4
C16-A	12500	500	4740	2.4	12.0	0.0	10.6	500	0.0	12	1.9	**11**	1920	11	0.9
C16-B	12500	500	4740	2.4	12.0	0.0	11.5	503	0.0	12	3.5	**11**	1758	11	13.8
C17-A	12500	498	4694	2.4	19.0	0.0	11.2	620	0.0	19	2.9	**18**	549	18	1.9
C17-B	12500	498	4694	2.3	18.2	0.4	12.7	1951	76.7	18	2.1	18	434	18	1.4
C18-A	12500	469	4569	2.6	112.4	0.7	24.1	7446	6.7	112	2.1	**111**	3990	111^{+}	—
C18-B	12500	465	4538	2.9	115.0	0.7	26.2	8361	6.7	116	219.5	**113**	3262	113^{+}	—
C19-A	12500	430	3982	2.9	146.2	0.4	17.9	5402	80.0	146	2.3	146	3928	146	0.6
C19-B	12500	416	3867	2.8	149.0	0.6	15.8	4035	0.0	147	3.0	**146**	3390	146	0.6
C20-A	12500	241	1222	6.1	266.0	0.0	7.3	598	100.0	266	0.2	266	4311	266	0.0
C20-B	12500	133	563	5.0	267.0	0.0	5.2	500	100.0	267	0.1	267	3800	267	0.1
D1-A	1250	231	440	4.9	18.0	0.0	3.1	500	100.0	18	0.0	18	6	18	0.0
D1-B	1250	233	443	4.9	106.0	0.0	3.8	1950	100.0	106	0.1	106	257	106	0.0
D2-A	1250	257	481	4.9	50.0	0.0	3.5	500	100.0	50	0.1	50	7	50	0.0
D2-B	1250	264	488	4.9	218.3	1.0	7.3	4157	93.3	**218**	0.1	228	486	218	0.0
D3-A	1250	301	529	5.5	807.0	0.0	7.4	500	100.0	807	0.1	807	734	807	0.1
D3-B	1250	372	606	6.3	1516.2	1.3	51.0	15976	0.0	**1509**	0.6	1510	2184	1509	0.3
D4-A	1250	311	541	5.6	1203.8	0.4	10.4	974	16.7	1203	0.3	1203	1263	1203	0.3
D4-B	1250	387	621	7.2	1885.2	2.0	49.6	9671	0.0	1881	11.0	1881	2233	1881	1.3
D5-A	1250	348	588	7.6	2157.0	0.0	29.1	1963	100.0	2157	3.1	2157	3352	2157	8.8
D5-B	1250	411	649	11.5	3137.7	0.9	65.1	7316	0.0	3135	2.2	3135	2555	3135	0.4
D6-A	2000	740	1707	14.4	18.0	0.0	7.7	500	100.0	18	0.3	18	20	18	0.1
D6-B	2000	741	1708	14.7	72.6	0.8	10.5	1192	0.0	71	0.5	**70**	702	67	0.9
D7-A	2000	734	1705	11.3	50.0	0.0	8.2	500	100.0	50	0.3	50	195	50	0.1
D7-B	2000	736	1707	11.4	105.0	0.0	9.5	520	0.0	105	0.3	105	711	103	0.1
D8-A	2000	764	1738	11.7	755.5	0.5	19.1	2788	50.0	755	15.6	755	1727	755	41.8
D8-B	2000	778	1757	12.3	1045.7	3.9	123.8	36313	0.0	**1037**	1013.4	1038	3175	1036	2.8
D9-A	2000	752	1716	17.9	1074.7	1.0	52.1	13718	0.0	1075	354.5	**1072**	4109	1070^{+}	—
D9-B	2000	761	1724	20.9	1436.4	3.0	151.2	31361	0.0	1420	1769.6	1420	2754	1420	4539.6
D10-A	2000	694	1661	14.6	1674.4	1.4	122.2	21289	0.0	1671	9.0	1671	4193	1671	2.2
D10-B	2000	629	1586	18.5	2089.8	2.1	107.3	14598	0.0	2079	44.1	2079	2644	2079	4.1
D11-A	5000	986	4658	27.7	18.0	0.0	15.4	500	100.0	18	1.8	18	540	18	0.5
D11-B	5000	986	4658	23.6	29.0	0.0	17.4	814	100.0	**29**	2.0	30	1280	29	4.7
D12-A	5000	991	4639	23.1	42.0	0.0	13.9	500	100.0	42	2.3	42	844	42	13.2
D12-B	5000	991	4639	22.3	42.0	0.0	15.1	620	100.0	42	2.3	42	687	42	0.4
D13-A	5000	966	4572	27.7	446.7	0.5	58.7	14308	0.0	445	1126.4	445	5047	445	5643.4
D13-B	5000	961	4566	28.0	491.7	1.9	97.2	22843	0.0	486	15.9	486	4288	486	2.6
D14-A	5000	946	4500	35.5	605.6	1.2	102.3	21486	0.0	602	34.2	602	6388	602*	—
D14-B	5000	931	4469	37.2	674.2	1.4	102.8	17746	0.0	665	3409.5	665	6178	664*	—
D15-A	5000	832	4175	47.1	1048.7	1.3	145.7	18343	0.0	1042	185.8	1042	7840	1042	12.8
D15-B	5000	747	3896	49.2	1114.7	0.8	95.6	11026	0.0	1108	117.0	1108	5220	1108	4.8
D16-A	25000	1000	10595	10.8	14.0	0.0	23.1	500	0.0	14	8.9	**13**	1397	13	24.8
D16-B	25000	1000	10595	10.8	13.3	0.4	26.4	1313	73.3	13	9.3	13	1043	13	42.0
D17-A	25000	999	10534	10.8	23.0	0.0	24.8	1983	100.0	23	9.5	23	3506	23	167.1
D17-B	25000	999	10534	10.7	23.0	0.0	23.7	948	100.0	23	10.2	23	2089	23	60.1
D18-A	25000	944	9949	11.7	220.8	0.7	81.4	19864	0.0	218	197.0	218	30044	218^{+}	—
D18-B	25000	929	9816	12.0	230.2	1.3	98.7	25585	0.0	224	25.2	224	36643	223	34.9
D19-A	25000	897	9532	12.4	317.7	2.7	87.6	18480	0.0	308	151.9	308	40955	306	1446.5
D19-B	25000	862	9131	13.1	317.8	2.2	81.9	17912	0.0	311	13.6	311	38600	310	62.8
D20-A	25000	488	2511	37.3	537.0	0.0	18.4	1036	0.0	536	1.0	536	28139	536	0.5
D20-B	25000	307	1383	32.9	537.0	0.0	12.7	1587	100.0	537	0.5	537	22104	537	0.1

Table 3 further illustrates the importance of using both, recombination and mutation, and that it is necessary to apply local improvement immediately after each variation operator. Shown are average results of 30 runs for the following three variants of the MA: In C+LI, new candidate solutions are created only by recombination followed by local improvement. M+LI applies always only mutation followed by local improvement. In C+M+LI, recombination and mutation are used, and local improvement is performed before a solution is inserted into the population. All strategy parameters were set identical as in the previous experiments with the only exception that in M+LI, the probability of applying mutation was $p_{\text{mut}} = 1$. The performance values of these variants can therefore directly be compared to those of the original MA in Table 2.

C+M+LI converged fastest, but the obtained solutions were in nearly all cases substantially poorer (1.7% of average gap over all instances) than those of the original MA (0.6% of average gap). This points out the particular importance of applying local improvement after *both* variation operators. C+LI, on the other side, generally needed much more evaluations and also more time to converge. Although its total running time hardly deviates form our original MA, the average gap obtained over all instances was 1.2 %. Finally, the worst results were obtained by running M+LI, with 2% of average gap, which clearly indicates the crossover's importance.

4 Conclusions and Future Research

Our results show that exact algorithms used as local improvement or post-optimization procedures can improve the performance of memetic algorithms. We conjecture that combining linear programming or integer linear programming methods with evolutionary algorithms as described in this paper can yield high quality solutions in short computation time also for other hard optimization problems.

In our future research, we want to combine memetic algorithms with a Branch & Cut approach for solving integer linear programs to obtain even better solutions. Since almost all the currently available benchmark instances are now solved to optimality within a rather short time, the frontier of tractable instances can be pushed further. Based on a real-world utility network design problem we plan to establish new sets of difficult benchmark instances to give new challenges to the community.

Table 2. Summarized results. Running times from Canuto et al. should be divided by 10 for comparison (cf. SPEC comparison). $\%\text{-}gap = (c(T) - OPT)/OPT \cdot 100\%$.

	Preprocessing			MA		MA+ILP		CRR		ILP
Group	$\lvert V'\rvert/\lvert V\rvert$ [%]	$\lvert E'\rvert/\lvert E\rvert$ [%]	t_{prep} [s]	%-*gap*	t [s]	%-*gap*	t [s]	%-*gap*	t [s]	t_{OPT} [s]
K	42.8	46.4	1.6	0.17	4.4	0.13	5.5	0.03	74.5	139.3
P	80.9	74.7	1.0	0.06	12.0	0.01	12.3	0.00	215.1	12.6
C	69.7	59.9	3.8	1.01	20.0	0.70	27.3	0.04	956.2	?
D	70.5	62.9	16.9	0.98	62.7	0.44	232.2	0.41	6834.6	?

Table 3. Average performance over 30 runs of different MA-variants, for `K, P, C` and `D` groups of PCSTP instances.

Grp.	C+LI					M+LI					C+M+LI				
	%-gap	σ	*t [s]*	*evals*	*sr [%]*	*%-gap*	σ	*t [s]*	*evals*	*sr [%]*	*%-gap*	σ	*t [s]*	*evals*	*sr [%]*
K	0.2	< 0.1	4.2	592	69.1	0.2	< 0.1	4.3	907	70.1	0.3	< 0.1	3.7	727	70.3
P	0.3	< 0.1	10.1	5076	46.1	0.3	0.1	11.6	7478	27.3	0.6	0.1	5.8	3040	19.1
C	2.2	0.1	17.4	6222	41.7	3.9	0.2	18.4	4264	24.6	2.4	0.2	11.0	1313	28.8
D	1.9	0.3	60.5	11582	27.4	3.7	0.9	64.7	9479	20.2	3.5	0.2	37.2	1697	18.2

References

1. D. Bienstock, M. X. Goemans, D. Simchi-Levi, and D. Williamson. A note on the prize collecting traveling salesman problem. *Math. Prog.*, 59:413–420, 1993.
2. 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*, 38:50–58, 2001.
3. C. W. Duin and A. Volgenant. Some generalizations of the Steiner problem in graphs. *Networks*, 17(2):353–364, 1987.
4. S. Engevall, M. Göthe-Lundgren, and P. Värbrand. A strong lower bound for the node weighted Steiner tree problem. *Networks*, 31(1):11–17, 1998.
5. M. Fischetti. Facets of two Steiner arborescence polyhedra. *Mathematical Programming*, 51:401–419, 1991.
6. M. X. Goemans. The Steiner tree polytope and related polyhedra. *Mathematical Programming*, 63:157–182, 1994.
7. M. X. Goemans and D. P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. In D. S. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pages 144–191. P. W. S. Publishing Co., 1996.
8. D. S. Johnson, M. Minkoff, and S. Phillips. The prize-collecting Steiner tree problem: Theory and practice. In *Proceedings of 11th ACM-SIAM Symposium on Discrete Algorithms*, pages 760–769, San Francisco, CA, 2000.
9. G. Klau, I. Ljubić, A. Moser, P. Mutzel, P. Neuner, U. Pferschy, and R. Weiskircher. A new lower bounding procedure for the prize-collecting Steiner tree problem. Technical Report TR-186-1-04-01, Vienna University of Technology, 2004.
10. G. Klau, I. Ljubić, P. Mutzel, U. Pferschy, and R. Weiskircher. The fractional prize-collecting Steiner tree problem on trees. In G. D. Battista and U. Zwick, editors, *ESA 2003*, volume 2832 of *LNCS*, pages 691–702. Springer-Verlag, 2003.
11. A. Lucena and M. Resende. Strong lower bounds for the prize-collecting Steiner problem in graphs. Technical Report 00.3.1, AT&T Labs Research, 2000.
12. K. Mehlhorn. A faster approximation for the Steiner problem in graphs. *Information Processing Letters*, 27:125–128, 1988.
13. P. Moscato. Memetic algorithms: A short introduction. In D. Corne and et al., editors, *New Ideas in Optimization*, pages 219–234. McGraw Hill, England, 1999.
14. G. R. Raidl and J. Gottlieb. On the importance of phenotypic duplicate elimination in decoder-based evolutionary algorithms. In S. Brave and A. S. Wu, editors, *Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference*, pages 204–211, Orlando, FL, 1999.
15. G. R. Raidl and B. A. Julstrom. Edge-sets: An effective evolutionary coding of spanning trees. *IEEE Trans. on Evolutionary Computation*, 7(3):225–239, 2003.
16. A. Segev. The node-weighted Steiner tree problem. *Networks*, 17:1–17, 1987.