Neighbourhood Searches for the Bounded Diameter Minimum Spanning Tree Problem Embedded in a VNS, EA, and ACO [*]

Martin Gruber
Institute for Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstraße 9–11/186-1, 1040 Vienna, Austria
gruber@ads.tuwien.ac.at

Jano van Hemert
National e-Science Institute, University of Edinburgh, 15 South College Street, Edinburgh EH8 9AA, UK
jano@vanhemert.co.uk

Günther R. Raidl
Institute for Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstraße 9–11/186-1, 1040 Vienna, Austria
raidl@ads.tuwien.ac.at

ABSTRACT

We consider the Bounded Diameter Minimum Spanning Tree problem and describe four neighbourhood searches for it. They are used as local improvement strategies within a variable neighbourhood search (VNS), an evolutionary algorithm (EA) utilising a new encoding of solutions, and an ant colony optimisation (ACO). We compare the performance in terms of effectiveness between these three hybrid methods on a suite of popular benchmark instances, which contains instances too large to solve by current exact methods. Our results show that the EA and the ACO outperform the VNS on almost all used benchmark instances. Furthermore, the ACO yields most of the time better solutions than the EA in long-term runs, whereas the EA dominates when the computation time is strongly restricted.

Categories and Subject Descriptors

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

General Terms

Algorithms

Keywords

bounded diameter minimum spanning tree problem, variable neighbourhood search, evolutionary computation, ant colony optimisation

[*]This work is supported by the RTN ADONET under grant 504438 and the Austrian Science Fund (FWF) under grant P16263-N04.

1. INTRODUCTION

The *bounded diameter minimum spanning tree* (BDMST) problem is a combinatorial optimisation problem appearing in applications such as wire-based communication network design when certain aspects of quality of service have to be considered, in ad-hoc wireless networks [3], and in the areas of data compression and distributed mutual exclusion algorithms [20, 6].

Given an undirected, connected graph $G = (V, E)$ of $n = |V|$ nodes and $m = |E|$ edges with associated costs $c_e \geq 0$, $e \in E$, the BDMST problem can be defined as follows: Determine a spanning tree $T = (V, E_T)$ with edge set $E_T \subseteq E$ whose diameter does not exceed a given upper bound $D \geq 2$ and whose total costs $\sum_{e \in E_T} c_e$ are minimal. This problem is known to be NP-hard for $4 \leq D < n - 1$ [10].

The *eccentricity* of a node v is the maximum number of edges on a path from v to any other node in the tree T. The *diameter* of T is the maximum eccentricity of all nodes, i.e., the largest number of edges on any path. The *centre* of T is the single node (in case the diameter of T is even) or the pair of adjacent nodes (if T's diameter is odd) of minimum eccentricity. A BDMST can also be interpreted as a spanning tree rooted at an unknown centre having its height H restricted to half of the diameter, i.e. $H = \lfloor \frac{D}{2} \rfloor$.

In this work we describe four neighbourhood structures for the BDMST problem which are utilised within a general *variable neighbourhood search* (VNS) approach [15, 14] for solving large instances heuristically. Furthermore, an *evolutionary algorithm* (EA) based on a newly developed level encoding and an *ant colony optimisation* approach are proposed. Both also make use of the neighbourhood searches in order to locally improve candidate solutions. All three metaheuristics are experimentally evaluated on a set of previously used benchmark instances consisting of complete Euclidean graphs with up to 1000 nodes. In comparison to previous work, the EA and the ACO were able to obtain new, significantly better solutions on almost all instances. In particular in long-term runs on larger instances, the ACO performs best with respect to solution quality, while the EA's results are better when the running time is strictly limited.

2. PREVIOUS WORK

Exact approaches for solving the BDMST problem mostly rely on flow-based multi-commodity mixed integer linear programming formulations [2, 11, 12]. A model based on lifted Miller-Tucker-Zemlin inequalities instead of network flows is presented in [9]. Recently, Gruber and Raidl [13] suggested a branch-and-cut algorithm based on a compact 0–1 integer linear programming formulation. However, due to the complexity of the BDMST problem the applicability of all these exact algorithms is, in practice, restricted to instances with less than 100 nodes when considering complete graphs.

Fast greedy construction heuristics for the BDMST problem are primarily based on the well-known minimum spanning tree (MST) algorithm by Prim. For example, the *one-time tree construction* (OTTC) by Abdalla et al. [1] starts with a tree consisting of a single, arbitrarily chosen node and repeatedly extends it by adding the cheapest available edge connecting a new node. To ensure the diameter bound the algorithm has to keep track of the eccentricities of each node already connected to the tree and discard infeasible edges; a relatively time-consuming update procedure is required. In contrast to Prim's MST algorithm the choice of the node to start with has a crucial impact on the generated tree and its costs.

Julstrom [17] modified this approach to start from a predetermined centre. This simplifies the algorithm significantly since the diameter constraint can be replaced by restricting the height of the generated tree to $\lfloor \frac{D}{2} \rfloor$. This *centre-based tree construction* (CBTC) runs for one chosen centre in time $O(n^2)$ while OTTC requires $O(n^3)$. Although this approach yields relatively good results on random instances, its behaviour is in general too greedy for Euclidean graphs. In this case a randomised version of CBTC, called *randomised centre-based tree construction* (RTC) [19], leads to significantly better results. It chooses the centre as well as the order in which all other nodes are appended at random, but again connects them with the cheapest possible edges. Other construction heuristics, for example a modification of Kruskal's MST algorithm, for the related height-constrained MST problem, can be found in [7].

Beside these greedy construction heuristics, also metaheuristics were developed for the BDMST problem in order to obtain better results. Raidl and Julstrom [19] presented an EA employing a direct edge-set encoding and four variation operators being able to produce new candidate solutions in almost $O(n)$ expected time. In [18] the same authors suggested an EA using a permutation representation, which determines the order in which the nodes are appended by an RTC-like decoding heuristic. This approach leads to better solutions, but with the drawback of longer running times for large instances since decoding a chromosome requires $O(n^2)$ time. Another EA based on random-keys has been proposed in [16]. A comparison to the permutation-coded approach indicated a similar performance. Recently in [14], Gruber and Raidl described a variable neighbourhood search approach involving four different neighbourhood structures, which outperformed all the so-far proposed EAs.

In the following, we will recapitulate this VNS and its four neighbourhoods and describe how they can be searched efficiently. The subsequent sections introduce a new EA and ACO which are based on these concepts.

node i	1	2	3	4	5	6	7	8	9
$lev(i)$	0	2	2	2	1	2	1	2	1

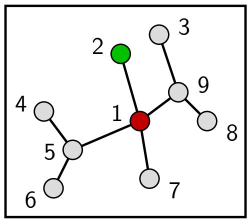


Figure 1: Decoding a level vector (Euclidean distances, $D = 4$).

3. SEARCH NEIGHBOURHOODS

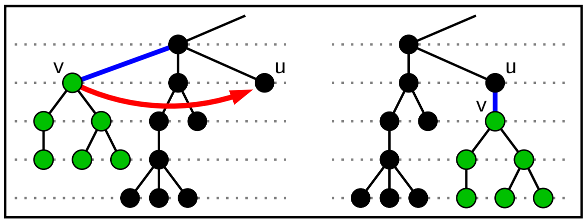
All our metaheuristics for the BDMST problem make use of one or more neighbourhood searches, which always only consider feasible solutions. For an efficient implementation, we represent a solution as an outgoing arborescence, i.e., a directed tree rooted at a centre, using the following data structures:

- an array *pred* containing for each node $v \in V$ its direct predecessor in the directed path from the centre to it, respectively `NULL` in case v is a centre node;
- for each node $v \in V$ a list $succ(v)$ of all its direct successors; for a leaf this list is empty;
- an array *lev* storing the level for each node $v \in V$, which is the length of the path from the centre to v;
- for each level $l = 0, \ldots, H$ a list V_l of all nodes at level l.

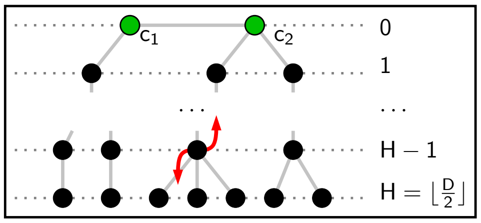
Out of the four neighbourhoods, the first two are based on the tree structure defined by the relationship between predecessors and successors. For the two remaining neighbourhoods we change the representation of a solution and therefore the search space: Instead of the predecessor respectively successor information, the level a node is assigned to is of main interest. Given the level $lev(v)$, $\forall v \in V$, it is straight-forward to derive an optimal bounded diameter spanning tree with respect to *lev*: To each non-centre node v we assign a least-cost predecessor from $V_{lev(v)-1}$.

In order to obtain even better solutions we go one step further and relax the meaning of "level" in this decoding procedure: For a node at level l, any node at a level smaller than l (not just $l-1$) is allowed as predecessor, and an overall cheapest connection is chosen. In case of ties a node of minimum level is selected. See Figure 1 for an example. Note in particular that node 2 has level 2 and is connected to the centre node 1 at level 0 since this is the nearest node at a smaller level.

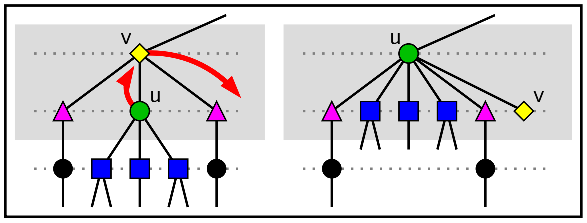
Algorithm 1 shows this decoding in a more detailed pseudo-code. In order to find the least-cost predecessor of a node i as quickly as possible, we use the following strategy: Only if the number of nodes assigned to a level smaller than $lev(i)$ is less than a threshold δ, we scan the lists V_0 to $V_{lev(i)-1}$ for the cheapest connection. Otherwise we make use of a precomputed nearest neighbour list for node i, which



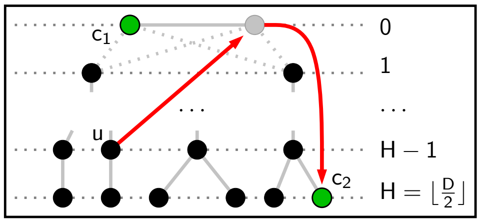
(a) Arc exchange neighbourhood.



(c) Level change neighbourhood.



(b) Node swap neighbourhood.



(d) Centre exchange level neighbourhood.

Figure 2: The four neighbourhood types defined for the BDMST problem.

Algorithm 1: Decoding a level vector *lev*.

```
1 for every node $v \in V$ do
2     if $\sum_{l=0}^{lev(v)-1} |V_l| < \delta$ then
3         search level lists $V_l, l = 0 \ldots lev(v) - 1$, for the
          cheapest predecessor $p$ for node $v$
4     else
5         search sorted nearest neighbour list of $v$ for the
          first node $p$ assigned to a level $< lev(v)$
6     $pred(v) \longleftarrow p$;
```

contains all nodes adjacent to i in increasing edge cost order. The first node in this nearest neighbour list at a level less than $lev(i)$ is chosen as predecessor for node i. In preliminary experiments $\delta \approx 0.1 \cdot n$ turned out to be a good choice in our implementation.

In the following we consider the four neighbourhoods in detail.

3.1 Arc Exchange Neighbourhood

The arc exchange neighbourhood of a current solution T consists of all feasible trees differing from T in exactly a single arc (directed edge). The associated move can be interpreted as disconnecting some sub-tree and reconnecting it at another feasible place, see Figure 2-a.

This neighbourhood consists of $O(n^2)$ solutions. A single neighbour can be evaluated in constant time when only considering cost differences. We ensure that the diameter constraint is not violated by predetermining for each node $v \in V$ the height $h(v)$ of the sub-tree rooted at it; feasible candidates for becoming new predecessor of a node v after disconnecting it are all nodes at levels less than or equal to $H - h(v) - 1$. Under this conditions, the total time for examining the whole neighbourhood in order to identify the best move is in $O(n^2)$.

3.2 Node Swap Neighbourhood

This neighbourhood focuses on the relationship between nodes and their set of direct successors. A neighbouring solution is defined as a tree in which a node v and one of its direct successors u exchange their positions within the tree: As illustrated in Figure 2-b node u becomes predecessor of v and successor of v's former predecessor. While node u can keep its successors (u is moved to a smaller level), the successors of v are reconnected to u in order to always ensure feasibility with respect to the diameter constraint in an easy way.

In contrast to arc exchange, a move in this neighbourhood can result in several new connections. Nevertheless, the whole neighbourhood has only size $O(n)$ and it can be efficiently examined in time $O(n \cdot d_{\max})$, where $d_{\max}$ is the maximum degree of any node in the current tree.

3.3 Level Change Neighbourhood

In the level change neighbourhood an adjacent solution is reached by incrementing or decrementing the level of exactly one node v at level $1 \leq lev(v) \leq H - 1$ and $2 \leq lev(v) \leq H$, respectively, and reapplying the decoding procedure presented in Algorithm 1; see Figure 2-c.

The size of this neighbourhood is $O(n)$ and an exhaustive examination can be implemented in time $O(n^2)$. Incremental evaluation speeds up the computation substantially in practice but does not reduce this worst-case time complexity.

If a node v at level l is connected to a predecessor u assigned to a level smaller than $l - 1$, as it is allowed by our decoding procedure, it is advantageous to reduce $lev(v)$ further by consecutive moves until $lev(v) = lev(u)+1$ because v can act as potential predecessor for more nodes. This is accomplished by accepting decrement moves even if they have no immediate impact on the objective value.

3.4 Centre Exchange Level Neighbourhood

The level change neighbourhood never affects the centre node(s). In order to fill this gap, we use the centre exchange level neighbourhood. It replaces exactly one centre node by any non-centre node u. The replaced centre node is set to level H, maximising the number of potential predecessors, see Figure 2-d.

To better exploit the potential of such a centre exchange a local improvement step is immediately appended: As long as there exists a node w whose predecessor v has level $lev(v) < lev(w) - 1$, we assign node w to level $lev(v) + 1$. Following such a reduction, node w can serve as potential predecessor for a larger number of other nodes and – as a consequence – cheaper connections might be enabled.

Restricting the exchange to exactly one centre node leads to a neighbourhood size of $O(n)$. A local improvement step requires in the worst-case time $O(n^2)$, yielding a total time complexity of $O(n^3)$ for evaluating the whole neighbourhood.

4. METHODOLOGIES

4.1 Variable Neighbourhood Search

The framework follows the general VNS scheme as proposed in [15] using *variable neighbourhood decent* (VND) as local search strategy.

An initial solution is created by one of the fast greedy construction heuristics. In our implementation we repeatedly applied RTC until no new improved solution has been obtained within the last n repetitions. Within VND we always use a best improvement strategy, i.e. each neighbourhood is completely explored and the best move is performed as long as it yields an improvement. The following order of neighbourhoods has proven to be successful: First, whole sub-trees are moved within the solution (arc exchange), afterwards the arrangement of nodes and their direct successors is considered (node swap). Then the usually more time consuming level based neighbourhoods are applied: The best centre with respect to the centre exchange level neighbourhood is determined, and finally the levels the non-centre nodes are assigned to are refined by means of the level change neighbourhood.

When performing a local search using a single neighbourhood and following a best improvement strategy, it is sometimes possible to store information during the exploration of this neighbourhood allowing a faster incremental search for the successive best move. We implemented such a scheme for the node swap and the level change neighbourhoods. To benefit from this advantage and in contrast to standard VND, we do not switch back to the first neighbourhood immediately after an improvement, but continue the local search within the same neighbourhood until a local optimum is reached. Only then we restart our search with the first neighbourhood in order to exploit further possible improvements.

Our general VNS framework is shown in Algorithm 2. Since VND always yields a solution that is locally optimal with respect to all used neighbourhoods, it makes usually no sense to shake a solution in the VNS performing only a single random move in one of these neighbourhoods. Therefore, shaking is performed by applying k random moves within one of the four neighbourhoods chosen at random, with k running from $k_{\text{start}} \geq 2$ to k_{max}.

Algorithm 2: VNS for the BDMST

```
1  create initial solution using RTC heuristic;
2  number of shaking moves $k \longleftarrow k_{start}$;
3  while termination condition not met do
4      perform VND with best improvement strategy:
       begin
5          neighbourhood $l \longleftarrow 1$;
6          while $l \leq 4$ and time limit not reached do
7              switch $l$ do perform local search to a local
               optimum using neighbourhood
8                  case 1 : arc exchange;
9                  case 2 : node swap;
10                 case 3 : centre exchange level;
11                 case 4 : level change;
12             if solution improved and $l \neq 1$ then
               $l \longleftarrow 1$  else  $l \longleftarrow l + 1$;
13     end
14     if best solution improved or $k \geq k_{max}$ then
       $k \longleftarrow k_{start}$  else  $k \longleftarrow k + 1$;
15     choose neighbourhood at random and shake
       currently best known solution using $k$ moves;
```

In case the centre exchange level neighbourhood has been chosen for the shaking process we make an exception and use a combination of it and the level change neighbourhood because iterated centre exchange moves alone cannot gain the desired larger variation: The first $1 + D \bmod 2$ (the number of centre nodes) moves are executed within the centre exchange level neighbourhood, and for the remaining $k - (1 + D \bmod 2)$ shaking moves we switch to the level change neighbourhood.

4.2 Evolutionary Algorithm

The evolutionary algorithm uses a novel representation, based on the concepts of the level neighbourhoods, to loosely represent solutions to the BDMST problem. It then relies on a decoding mechanism to transform this representation into a specific tree. A BDMST is represented as a string of levels, i.e., each gene represents the level of a corresponding node in the directed tree. This level information on itself does not completely encode a tree, and therefore, we use the Algorithm 1 to derive the predecessor information, i.e., a specific tree. This predecessor information is also stored in the individual and local improvement is applied.

The local improvement consists of applying only the arc exchange neighbourhood search from Section 3.1. If an improvement can be made, which is the case more often than not, the node level representation and fitness value are set according to the new improvement. The arc exchange neighbourhood is chosen to perform the local improvement as it is computationally relatively cheap in practice and it complements the genetic operators, which provide new points in the search space by changing level information, and especially, by moving the centres around.

We decided not to include the other three neighbourhood searches. This decision is based on preliminary experiments where we found including them almost always extremely increased the necessary running times for converging to high-quality solutions. These experiments consisted of including the three neighbourhood searches with equal probability to the arc exchange neighbourhood search, including each of them with the small probability of 0.10, and including each of them with a very small probability of 0.01.

An important aspect of the representation is that an individual is allowed a limited number of centre nodes only. These nodes have a level of zero. If the BDMST problem has an even diameter, one node is allowed, if it is odd, two nodes form the center. When generating the initial population, genes are assigned levels of one to the maximum height of the tree. Thereafter, one or two nodes are randomly assigned to level zero, depending on the parity of the problem.

The genetic operators are of a simple, straight-forward nature, with the exception where they cater for the centre nodes. A uniform crossover is performed where for each gene a level is chosen with equal probability from the two parents. The locations of the centre nodes of both parents are put into a pool. Then one or two are selected randomly to represent the centre nodes of the offspring. The remaining locations are checked in the offspring, if they contain centre nodes inherited from the uniform crossover, then these genes are overwritten with a random level between one and the maximum tree height. The mutation operator generates a new level for each gene with a probability of $1/n$, skipping centre nodes. Afterwards, the centre nodes are swapped with other nodes selected randomly with equal probability.

A steady-state evolutionary model with binary tournament selection for choosing the parents is used. A new candidate solution always replaces the worst solution in the population, with one exception: duplicate individuals are not allowed to enter the population. Crossover and mutation are always applied, as well as local improvement.

4.3 Ant Colony System

In our ant colony system (ACS, Dorigo and Gambardella [8]) we also exploit the idea that a solution to the BDMST problem can be derived from an assignment of nodes to levels $0 \ldots H$ by Algorithm 1.

Therefore we make use of a $n \times (H+1)$ pheromone matrix τ where each value $\tau_{i,l}$ denotes the pheromone value for a node i at level l. The pheromone matrix is uniformly initialised with $\tau_{i,l} = (n \cdot T_0)^{-1}$, with T_0 being the objective value of a heuristic solution to the BDMST problem computed using one of the presented greedy construction heuristics, e.g. RTC for Euclidean instances (see Section 2).

To derive a valid solution we have to restrict the number of centre nodes, so they are chosen first: A node i is selected in proportion to its pheromone value $\tau_{i,0}$ at level 0, i.e., with a probability

$$P_{i,0} = \frac{\tau_{i.0}}{\sum_{j \in V} \tau_{j.0}}.$$

In case the diameter is odd, the second centre node is selected analogously from all remaining nodes.

After the centre has been determined all other nodes are assigned to the available levels $1 \ldots H$ independently from each other. The probability for a node i to be set to level l is defined similar as for the centre nodes, namely

$$P_{i,l} = \frac{\tau_{i.l}}{\sum_{l'=1}^{H} \tau_{i.l'}}.$$

Note in particular that these probabilities do not include any local heuristic component.

After each node has been assigned to a level, a corresponding BDMST is derived using Algorithm 1, where every node at a level ≥ 1 is connected to the cheapest available predecessor at any smaller level. Afterwards, this tree is locally improved using a VND exploiting only the arc exchange and node swap neighbourhoods. The ideas of the level based neighbourhoods are already captured in the construction and decoding phase. In practice, the use of these neighbourhoods does not lead to a further improvement of the solution quality, but only significantly increases running time.

After each ant has built a BDMST the pheromone evaporation $\tau_{i.l} \leftarrow (1-\rho) \cdot \tau_{i.l}$, where $\rho \in [0,1)$ represents the pheromone decay coefficient, is triggered on all entries of the pheromone matrix. Only the best ant of an iteration is allowed to deposit pheromone: if node i is assigned to level l we set $\tau_{i.l} \leftarrow \tau_{i.l} + \rho \cdot \frac{1}{T^+}$ with T^+ being the objective value of the best BDMST in the current iteration.

5. EXPERIMENTS AND RESULTS

5.1 Experimental Setup

We will now experimentally compare the three methodologies from Section 4 for the BDMST problem. As in [19, 16] we use instances from Beasley's OR-Library [5, 4], which were originally proposed for the Euclidean Steiner tree problem. These instances contain coordinates of points in the unit square, and the Euclidean distances between any pair of points are the edge costs. For our experiments we used the first five instances of each size $n = 100$, 250, 500, and 1000. The maximum diameters were set to 10, 15, 20, and 25, respectively. All tests were performed on a Pentium®4 2.8 GHz PC using Linux as operating system.

For the different approaches the following parameters were used: For the EA a population size of 100 was chosen. The number of artificial ants in the ACO was 25, the value for the pheromone decay coefficient ρ was studied in depth by extensive preliminary tests and was set in dependence on the size of the instance, namely $\rho = 0.003$, 0.005, 0.006, and 0.008 for the 100, 250, 500, and 1000 node instances. The VNS also used different values for the shaking parameters k_{start} and $k_{\max}$ based on the instance size: they were set to (3,15), (4,20), (5,25), and (5,50), respectively.

Regarding the termination condition we performed two different series of experiments: long-term and short-term runs. For the long-term runs we used a CPU time limits of 2000, 3000, and 4000 seconds for the 100, 250, and 500 node instances. In addition, in case of the VNS and ACO a run was also terminated after 1000 iterations without further improvement of the best solution, since in this situation these two algorithms – in contrast to the EA – can be considered to have converged and further improvements are extremely unlikely.

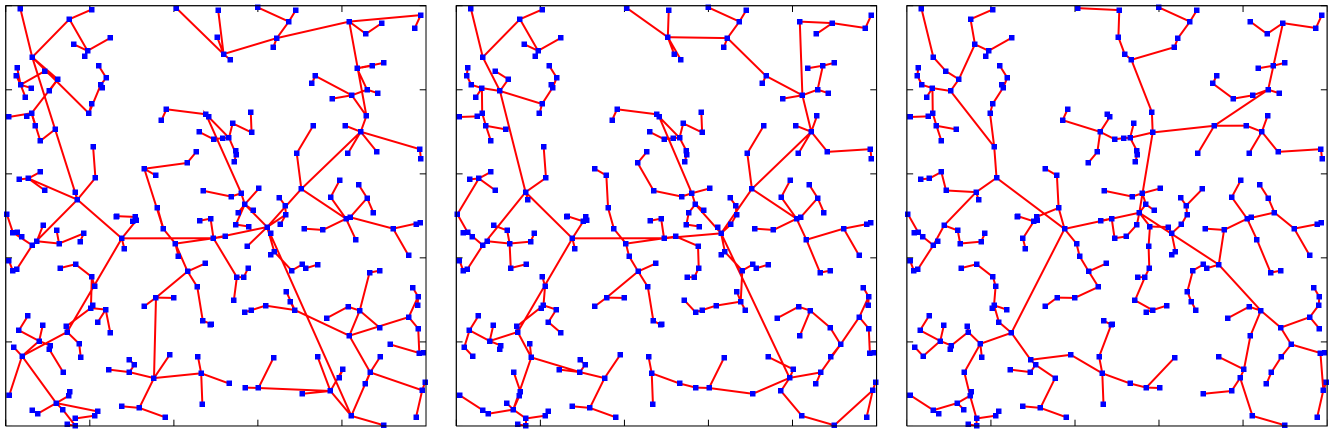
All statistical evaluations are based on 30 (VNS) respectively 50 (EA, ACO) independent runs for each instance.

5.2 Results

Table 1 shows the results for long runs on instances with 100, 250, and 500 nodes, where the main focus lies on the quality of the built tree. Listed are best and mean objective values, the corresponding standard deviations and the average times to identify the finally best solutions for each instance and the three metaheuristics under consideration.

Table 1: Final objective values of long-term runs on Euclidean instances.

Instance			VNS				level-encoded EA				ACO			
n	D	nr	best	mean	stdev	sec.	best	mean	stdev	sec.	best	mean	stdev	sec.
100	10	1	**7.759**	7.819	0.03	37.35	7.760	7.785	0.03	678.70	**7.759**	**7.768**	0.02	27.78
		2	7.852	7.891	0.03	41.52	**7.849**	**7.860**	0.02	734.65	7.850	7.864	0.01	25.10
		3	**7.904**	7.962	0.04	38.66	**7.904**	7.964	0.04	897.58	7.907	**7.943**	0.04	28.48
		4	7.979	8.046	0.03	34.27	**7.977**	8.008	0.03	732.83	7.979	**8.000**	0.01	38.24
		5	8.165	8.203	0.03	39.31	**8.164**	8.176	0.03	410.17	**8.164**	**8.170**	0.00	25.45
250	15	1	12.301	12.430	0.05	1584.31	12.280	12.377	0.05	1992.70	**12.231**	**12.280**	0.02	174.17
		2	12.024	12.171	0.06	1678.90	12.054	12.156	0.06	1969.42	**12.016**	**12.038**	0.01	156.71
		3	12.041	12.112	0.04	1309.21	12.026	12.095	0.04	1897.87	**12.004**	**12.021**	0.01	145.29
		4	12.507	12.615	0.06	1572.39	12.487	12.594	0.05	1742.48	**12.462**	**12.486**	0.01	159.41
		5	12.281	12.423	0.07	1525.39	12.319	12.423	0.06	1712.16	**12.233**	**12.288**	0.04	211.11
500	20	1	16.974	17.129	0.07	3718.54	16.866	16.967	0.06	2609.28	**16.778**	**16.850**	0.03	906.17
		2	16.879	17.052	0.07	3762.02	16.764	16.858	0.05	2472.59	**16.626**	**16.699**	0.03	1012.91
		3	16.975	17.148	0.07	3849.42	16.856	16.977	0.05	2808.15	**16.792**	**16.844**	0.03	1069.84
		4	16.992	17.166	0.06	3687.97	16.943	17.040	0.06	2837.81	**16.796**	**16.923**	0.04	1010.91
		5	16.572	16.786	0.07	3693.13	16.501	16.590	0.05	2294.43	**16.421**	**16.456**	0.02	947.26



(a) Best RTC solution after building 100 trees without further improvement;
objective value: 15.149.

(b) RTC solution from (a) improved by VND exploiting all four neighbourhoods;
objective value: 13.396.

(c) Best solution found so far for this instance by the ACO;
objective value: 12.231.

Figure 3: Euclidean instance number 1 with 250 nodes, $D = 15$.

Best results over the three algorithms are printed in bold. The instances with $n = 100$ seem too small to provide a proper comparison; each algorithm finds the best results for some of the instances. Furthermore, the objective values of the solutions over the three algorithms are rather similar and do not allow any conclusion to be drawn.

On all larger instances with 250 and 500 nodes, the ACO clearly outperforms VNS and the EA. In fact, the ACO's observed mean objective values are never worse than the single best solutions identified by one of the other approaches. Furthermore, the ACO's standard deviations are smallest indicating a higher reliability of finding high-quality solutions.

Comparing VNS with the level-encoded EA on the 250 and 500 node instances, the mean values of the EA are always smaller than those of VNS with exception of the fifth instance with 250 nodes, where they are equal.

After verifying our data are normally distributed, we performed unpaired t-tests between each pair of algorithms for each problem instance. With a significance level of 1%, the difference in results between the EA and the VNS are all significant with the exceptions of instance 2 to 5 for $n = 250, D = 15$, and instance 3 for $n = 100, D = 10$. The differences between the ACO and the VNS are all significant, except for instance 3 for $n = 100, D = 10$. All differences between the EA and the ACO are significant, except for instances 2,4, and 5 for $n = 100, D = 10$.

When looking at the average times until the best solutions have been found the ACO was in almost all cases substantially faster than VNS and the EA. Furthermore, on smaller instances the VNS found its final solutions in shorter time than the EA; on the largest considered instances the situation was vice versa.

Table 2: Final objective values of short-term runs on Euclidean instances.

Instance			limit	VNS				level-encoded EA				ACO			
n	D	nr	sec.	best	mean	stdev	sec.	best	mean	stdev	sec.	best	mean	stdev	sec.
500	20	1	50	17.753	18.108	0.12	46.41	**16.573**	**16.760**	0.16	37.94	17.594	17.751	0.06	41.29
		2		17.688	17.966	0.10	44.70	**16.826**	**17.014**	0.11	41.06	17.403	17.583	0.05	40.33
		3		17.799	18.114	0.10	46.23	**16.947**	**17.192**	0.13	43.15	17.653	17.756	0.05	39.66
		4		17.930	18.161	0.11	45.38	**16.957**	**17.085**	0.08	39.18	17.647	17.793	0.05	41.41
		5		17.464	17.863	0.12	45.94	**17.055**	**17.245**	0.13	39.54	17.331	17.438	0.05	40.95
500	20	1	500	17.290	17.460	0.08	476.22	**16.534**	**16.641**	0.07	340.34	17.017	17.150	0.07	485.57
		2		17.215	17.373	0.08	480.87	**16.808**	**16.902**	0.05	320.84	16.864	17.072	0.08	478.47
		3		17.252	17.464	0.05	476.33	**16.886**	**17.017**	0.06	319.09	17.094	17.259	0.07	479.17
		4		17.318	17.514	0.07	476.80	**16.923**	**17.036**	0.06	316.33	17.070	17.277	0.08	472.57
		5		16.932	17.139	0.09	473.82	17.007	17.105	0.06	288.66	**16.613**	**16.791**	0.08	479.93
1000	25	1	100	25.850	26.188	0.13	75.40	**24.831**	**25.019**	0.10	92.06	25.246	25.437	0.07	81.42
		2		25.501	25.981	0.17	68.30	**24.890**	**25.159**	0.10	89.29	25.092	25.239	0.07	80.17
		3		25.340	25.705	0.09	62.33	25.021	25.338	0.14	92.27	**24.870**	**25.007**	0.06	73.96
		4		25.562	26.128	0.17	73.89	**25.133**	25.524	0.12	92.17	25.329	**25.450**	0.06	76.56
		5		25.504	25.826	0.15	74.75	25.493	25.675	0.08	89.18	**24.884**	**25.153**	0.07	79.90
1000	25	1	1000	25.177	25.572	0.14	905.50	**23.434**	**23.573**	0.08	565.38	24.842	25.033	0.07	812.78
		2		25.015	25.342	0.14	930.04	**23.464**	**23.668**	0.08	561.49	24.634	24.834	0.06	847.79
		3		24.816	25.086	0.11	956.06	**23.635**	**23.793**	0.08	524.21	24.498	24.619	0.06	838.68
		4		25.289	25.572	0.11	928.97	**23.787**	**23.962**	0.09	602.30	24.993	25.091	0.06	793.41
		5		25.026	25.254	0.12	935.85	**23.837**	**23.982**	0.10	516.74	24.571	24.732	0.06	844.67

Figure 4 shows the mean objective value over time for multiple runs of the VNS, EA and ACO on instance number 2 with 500 nodes and a diameter of 20. The bottom of the chart displays the distributions of running times required to identify the best solution of a run, where mean running times are indicated by a vertical line each.

In our short-term experiments, we tested the approaches under CPU-time limits of 50 and 500 seconds for the 500 node instances, as well as 100 and 1000 seconds for the 1000 node instances. Table 2 shows results of these short-term runs. Here we see that roles are reversed, as in most cases the mean results of the EA are better than those of the ACO. Both, the EA and the ACO, almost always outperform the VNS. Interesting to note is that, with only a few exceptions, the mean results of the EA are already better than the best results found by the VNS, and this also holds true for the mean values of the ACO over the best of the VNS. Furthermore, when looking at the average computation times to identify the finally best solutions, the EA is usually faster than the ACO and VNS.

The objective value differences between all algorithms are statistically significant on an error-level of 1%, except for the EA and ACO on instances 3 and 5 for $n = 1000$, $D = 25$ with a time limit of 100 seconds.

In this comparison, we used for the short-term runs the same strategy parameters as for the high-quality experiments, which proved to be robust in excessive preliminary tests. However, there would be different possibilities to tune the algorithms to better perform when time is short. For example the VNS can omit the very time consuming centre exchange level neighbourhood; the idea of the level representation is still captured with the level change neighbourhood. The population size of the EA can be reduced, as well as the number of ants in the ant colony. In addition, a higher pheromone decay coefficient ρ can be used to influence the convergence behaviour of the ACO.

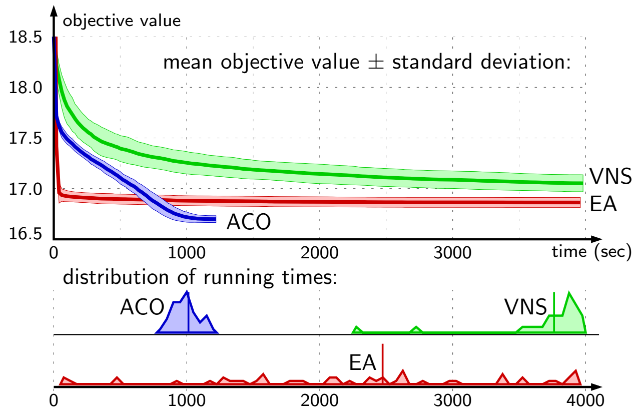


Figure 4: Objective value over time and running time distribution; long-term runs, $n = 500$, $D = 20$, instance 2.

Figure 3 shows exemplary solutions for the first 250 node instance with diameter bound $D = 15$ created by the iterated randomized tree construction heuristic (RTC), RTC followed by VND using all four neighbourhoods, and the ACO. Quality differences are visually clearly perceptible.

6. CONCLUSIONS

We have introduced two new methods for solving the Bounded Diameter Minimum Spanning Tree problem, which make use of neighbourhood searches that were developed for a variable neighbourhood search metaheuristics. The latter approach was until now the most successful in providing high-quality solutions for instances too large to be solved by exact methods. Our first new method is an evolutionary algorithm that encodes candidate trees only incompletely by node levels and uses a decoding procedure to complement solutions. Similarly, an ant colony optimisation algorithm is introduced where pheromone values are associated to the assignment of nodes to levels and the same decoding procedure is applied. In both metaheuristics, candidate spanning trees are further improved by a choice of neighbourhood searches.

We tested the two new methods on complete Euclidean benchmark instances with up to 1000 nodes and compared our results to those of variable neighbourhood search. Long-term runs were performed with the major aim to produce solutions of highest possible quality, while in short-term experiments, we investigated the performance under tight time restrictions.

Our results show that the evolutionary algorithm with the arc exchange neighbourhood search as local optimisation leads to significantly better results in runs with a tight limit on execution time when compared to the variable neighbourhood search and the ant colony optimisation algorithm. When running time is less critical, the tables turn, and it is the ant colony optimisation algorithm, also introduced here, that improves on the previous best known optima found by the variable neighbourhood search.

More study is required to accurately set parameters depending on the maximum time allowed for execution in order to maximise the potential of the algorithms. Furthermore, we anticipate performing experiments on problem instances where a specific structure is present, which should aid in better understanding both the strengths and weaknesses of each algorithm.

7. REFERENCES

[1] A. Abdalla, N. Deo, and P. Gupta. Random-tree diameter and the diameter constrained MST. *Congressus Numerantium*, 144:161–182, 2000.

[2] N. R. Achuthan, L. Caccetta, P. Caccetta, and J. F. Geelen. Computational methods for the diameter restricted minimum weight spanning tree problem. *Australasian Journal of Combinatorics*, 10:51–71, 1994.

[3] K. Bala, K. Petropoulos, and T. E. Stern. Multicasting in a linear lightwave network. In *IEEE Conference on Computer Communications*, pages 1350–1358, 1993.

[4] J. Beasley. OR-Library: Euclidean Steiner problem, 2005. `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/esteininfo.html`.

[5] J. E. Beasley. A heuristic for Euclidean and rectilinear Steiner problems. *European Journal of Operational Research*, 58:284–292, 1992.

[6] A. Bookstein and S. T. Klein. Compression of correlated bit-vectors. *Information Systems*, 16(4):387–400, 1991.

[7] A. E. F. Clementi, M. D. Ianni, A. Monti, G. Rossi, and R. Silvestri. Experimental analysis of practically efficient algorithms for bounded-hop accumulation in ad-hoc wireless networks. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, workshop 12*, volume 13, page 247.1, 2005.

[8] M. Dorigo and L. M. Gambardella. Ant colony system: A coopeartive learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[9] A. C. dos Santos, A. Lucena, and C. C. Ribeiro. Solving diameter constrained minimum spanning tree problems in dense graphs. In *Proceedings of the International Workshop on Experimental Algorithms*, volume 3059 of *LNCS*, pages 458–467. Springer, 2004.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.

[11] L. Gouveia and T. L. Magnanti. Network flow models for designing diameter-constrained minimum spanning and Steiner trees. *Networks*, 41(3):159–173, 2003.

[12] L. Gouveia, T. L. Magnanti, and C. Requejo. A 2-path approach for odd-diameter-constrained minimum spanning and Steiner trees. *Networks*, 44(4):254–265, 2004.

[13] M. Gruber and G. R. Raidl. A new 0–1 ILP approach for the bounded diameter minimum spanning tree problem. In L. Gouveia and C. Mourão, editors, *Proceedings of the 2nd International Network Optimization Conference*, volume 1, pages 178–185, Lisbon, Portugal, 2005.

[14] M. Gruber and G. R. Raidl. Variable neighborhood search for the bounded diameter minimum spanning tree problem. In P. Hansen, N. Mladenović, J. A. M. Pérez, B. M. Batista, and J. M. Moreno-Vega, editors, *Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search*, Tenerife, Spain, 2005.

[15] P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-heuristics, Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458. Kluwer Academic Publishers, 1999.

[16] B. A. Julstrom. Encoding bounded-diameter minimum spanning trees with permutations and with random keys. In K. Deb et al., editors, *Genetic and Evolutionary Computation Conference – GECCO 2004*, volume 3102 of *LNCS*, pages 1282–1281. Springer, 2004.

[17] B. A. Julstrom. Greedy heuristics for the bounded-diameter minimum spanning tree problem. Technical report, St. Cloud State University, 2004. Submitted for publication in the ACM Journal of Experimental Algorithmics.

[18] B. A. Julstrom and G. R. Raidl. A permutation-coded evolutionary algorithm for the bounded-diameter minimum spanning tree problem. In A. Barry, F. Rothlauf, D. Thierens, et al., editors, *in 2003 Genetic and Evolutionary Computation Conference's Workshops Proceedings, Workshop on Analysis and Desgn of Representations*, pages 2–7, 2003.

[19] G. R. Raidl and B. A. Julstrom. Greedy heuristics and an evolutionary algorithm for the bounded-diameter minimum spanning tree problem. In G. Lamont et al., editors, *Proceedings of the 2003 ACM Symposium on Applied Computing*, pages 747–752, New York, 2003. ACM Press.

[20] K. Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 7(1):61–77, 1989.