Variable Neighborhood Search for the Bounded Diameter Minimum Spanning Tree Problem

Martin Gruber[1], Günther R. Raidl[1] *

[1]Institute of Computer Graphics and Algorithms
Vienna University of Technology
Favoritenstraße 9–11/186-1, 1040 Vienna, Austria
{gruber|raidl}@ads.tuwien.ac.at

Abstract

The bounded diameter minimum spanning tree problem is an NP-hard combinatorial optimization problem with applications in various fields like communication network design. We propose a general variable neighborhood search approach for it, utilizing four different types of neighborhoods. They were designed in a way enabling an efficient incremental evaluation and search for the best neighboring solution. An experimental comparison on instances with complete graphs with up to 1000 nodes indicates that this approach consistently outperforms the so far leading evolutionary algorithms with respect to solution quality and computation time.

Keywords: bounded diameter minimum spanning tree, variable neighborhood search

1 Introduction

The *bounded diameter minimum spanning tree* (BDMST) problem is a combinatorial optimization problem appearing in various applications, for example in communication network design when certain aspects of quality of service have to be considered or in ad-hoc wireless networks [3]. It is also met in the areas of data compression or distributed mutual exclusion algorithms [16, 4].

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

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 described as follows: Find 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 $4 \leq D < n - 1$ [7].

The *eccentricity* of a node v is defined as 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 its nodes, thus, the largest number of edges on any path. The *center* 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 center having its height H restricted to half of the diameter, i.e. $H = \lfloor \frac{D}{2} \rfloor$.

In this work we will present a general *variable neighborhood search* (VNS) approach [11] based on four different types of neighborhoods for the BDMST problem. Our experiments on complete graphs with up to 1000 nodes illustrate that this algorithm consistently outperforms the so far leading (meta-)heuristics.

2 Previous Work

Exact approaches for solving the BDMST problem mostly rely on flow-based multi-commodity mixed integer linear programming formulations [2, 8, 9]. A model based on lifted Miller-Tucker-Zemlin inequalities instead of network flows is presented in [6]. Recently, we suggested a branch-and-cut algorithm based on a compact 0–1 integer linear programming formulation [10]. 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 (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 approach by Abdalla et al. [1], named *one-time tree construction* (OTTC), 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 must 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.

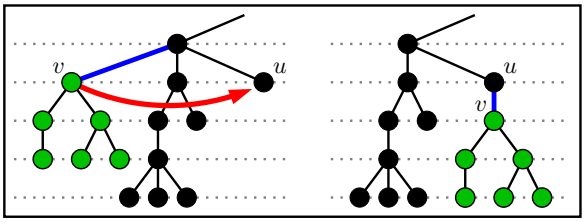
Julstrom [13] modified this approach to start from a predetermined center. This simplifies the algorithms significantly since the diameter constraint can be replaced by restricting the height of the generated tree to $\lfloor \frac{D}{2} \rfloor$. This *center-based tree construction* (CBTC) runs for one chosen center in time $O(n^2)$ while OTTC requires $O(n^3)$. Although this approach yields relatively good results on random instances its behavior is in general too greedy for Euclidean graphs. In this case a randomized version of CBTC, called *randomized center-based tree construction* (RTC) [15], leads to much better results. It chooses the center 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 [5].

Beside these greedy heuristics various evolutionary algorithms (EAs) were developed for the BDMST problem in order to obtain better results. Raidl and Julstrom [15] 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 [14] 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 [12]. A comparison to the permutation-coded approach indicated a similar performance.

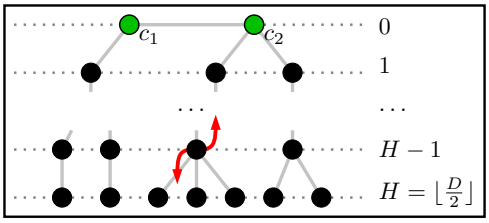
3 Neighborhood Types for the BDMST

Our VNS for the BDMST problem uses a total of four different types of neighborhoods, which always only consider feasible solutions. We represent a solution as an outgoing arborescence, i.e. a directed tree rooted at the center, using the following data structures:

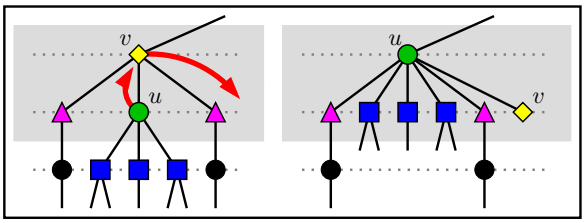
- An array *pred* containing for each node $v \in V$ its direct predecessor in the directed path from the center to it, respectively `NULL` in case v is a center node.
- For each node $v \in V$ a list $succ(v)$ of all its direct successors; for a leaf this list will be empty.



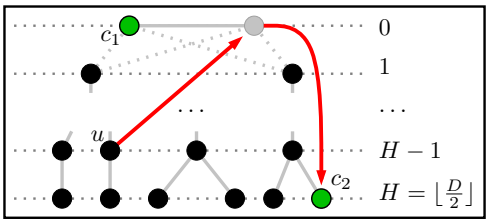
(a) Edge exchange neighborhood.



(c) Level change neighborhood.



(b) Node swap neighborhood.



(d) Center exchange level neighborhood.

Figure 1: The four neighborhood types defined for the BDMST problem.

- An array *lev* storing the level for each node $v \in V$, which is the length of the path from the center to v.
- For each level $l = 0, \ldots, H$ a list V_l of all its nodes.

3.1 Tree Structure Based Neighborhoods

The following two neighborhoods are based on the tree structure defined by the relationship between predecessors and successors in the tree.

3.1.1 Edge Exchange Neighborhood

The edge exchange neighborhood of a current solution T consists of all feasible trees differing from T in exactly a single (directed) edge. The associated move can be interpreted as disconnecting some subtree and reconnecting it at another feasible place, see Fig. 1-a.

This neighborhood consists of $O(n^2)$ solutions. A single neighbor 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 subtree 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 neighborhood in order to identify the best move is $O(n^2)$.

3.1.2 Node Swap Neighborhood

This neighborhood focuses on the relationship between nodes and their set of direct successors. A neighboring 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 Fig. 1-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 contrast to edge exchange, a move in this neighborhood can result in several new connections. Nevertheless the whole neighborhood 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.

3.2 Level Based Neighborhoods

In the two remaining neighborhoods not the predecessor respectively successor information but the level a node is assigned to is of main interest. Given the level information $lev(v)$, $\forall v \in V$, it is straight-forward to derive an optimal bounded diameter spanning tree with respect to lev: To each non-center 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.

3.2.1 Level Change Neighborhood

In the level change neighborhood 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 above decoding procedure; see Fig. 1-c.

The size of this neighborhood is $O(n)$ and an exhaustive examination can be implemented in time $O(n^2)$. Incremental evaluation speeds up the computation substantially 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 improved 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 instant impact on the objective value.

3.2.2 Center Exchange Level Neighborhood

The level change neighborhood never affects the center. In order to complement it, we introduce the center exchange level neighborhood. It replaces exactly one center node by any non-center node u. The replaced center node is set to level H, maximizing the number of potential predecessors, see Fig. 1-d.

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

Restricting the exchange to exactly one center node leads to a neighborhood 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 neighborhood.

4 The General Variable Neighborhood Search Framework

Our framework follows the general VNS scheme as proposed in [11] using *variable neighborhood decent* (VND) as local search strategy.

An initial solution is created by one of the fast greedy construction heuristics. In our implementation we applied RTC using the best solution from several runs. Within VND we always use a best improvement strategy, i.e. each neighborhood is completely explored and the best move is performed as long as it yields an improvement. The following order of neighborhoods has proven to

Algorithm 1: VNS for the BDMST

```
1  create initial solution using RTC heuristic;
2  number of shaking moves k ⟵ k_start;
3  while termination condition not met do
4      perform VND with best improvement strategy: begin
5          neighborhood l ⟵ 1;
6          while l ≤ 4 and time limit not reached do
7              switch l do perform local search to a local optimum using neighborhood
8                  case 1 : edge exchange;
9                  case 2 : node swap;
10                 case 3 : center exchange level;
11                 case 4 : level change;
12             if solution improved and l ≠ 1 then l ⟵ 1 else l ⟵ l + 1;
13     end
14     if best solution improved or k ≥ k_max then k ⟵ k_start else k ⟵ k + 1;
15     choose neighborhood at random and shake currently best known solution using k moves;
```

be successful: First, whole subtrees are moved within the solution (edge exchange), afterwards the arrangement of nodes and their direct successors is considered (node swap). Then the usually more time consuming level based neighborhoods are applied: The best center with respect to the center exchange level neighborhood is determined, and finally the levels the non-center nodes are assigned to are refined by means of the level change neighborhood.

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

Our general VNS framework is shown in Algorithm 1. Since VND always yields a solution that is locally optimal with respect to all used neighborhoods, it makes usually no sense to shake a solution in the VNS performing only one single random move in these neighborhoods. Therefore, shaking is performed by applying k random moves within one of the four neighborhoods chosen at random, with k running from $k_{start} \geq 2$ to k_{max}.

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

5 Computational Results

We will now experimentally compare our VNS implementation with state-of-the-art meta-heuristics for the BDMST problem. As in [15, 12] we use instances from Beasley's OR-Library [1] which have been 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 system using Linux 2.4.21 as operating system.

We compare our VNS approach with the leading evolutionary algorithms from [12] based on permutation and random-key representations. VNS uses a least-cost tree identified in multiple runs of RTC as initial solution: This construction heuristic is repeatedly performed until no better solution was obtained during the last n iterations. As stopping condition for VNS we used a combination of a CPU time limit (2000, 3000, and 4000 seconds for the 100, 250, and 500 node instances, respectively) and a maximum of 1000 consecutive applications of shaking without further improvement of the best solution. Depending on the problem size we also used different values k_{start} and k_{max} for shaking as indicated in Table 1.

Table 1 further lists for each instance the number of nodes, the maximum diameter, the instance number, and for each approach the best found solution, the mean, and the standard deviation of 50 (EAs) respectively 30 (VNS) independent runs. In addition, for VNS the mean times to find the best solutions are given.

The results are clear and consistent among all instances: VNS outperforms both EAs with respect to the best found solutions as well as with respect to the mean values. Sometimes, especially on larger instances, even the mean over all VNS runs is better than the overall best solutions identified

[1] `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/esteininfo.html`

Instance			permutation coded EA			random-key coded EA			VNS					
n	D	nr.	best	mean	stddev	best	mean	stddev	k_{start}	$k_{\max}$	best	mean	stddev	time (sec.)
100	10	1	7.818	7.919	0.07	7.831	7.919	0.05	3	15	**7.759**	**7.819**	0.03	37.35
		2	7.873	8.017	0.08	7.853	8.043	0.09			**7.852**	**7.891**	0.03	41.52
		3	7.990	8.139	0.08	7.982	8.137	0.09			**7.904**	**7.962**	0.04	38.66
		4	8.009	8.143	0.07	7.996	8.122	0.06			**7.979**	**8.046**	0.03	34.27
		5	8.193	8.335	0.08	8.198	8.313	0.08			**8.165**	**8.203**	0.03	39.31
250	15	1	12.440	12.602	0.08	12.448	12.580	0.08	4	20	**12.301**	**12.430**	0.05	1584.31
		2	12.237	12.432	0.10	12.222	12.393	0.10			**12.024**	**12.171**	0.06	1678.90
		3	12.117	12.282	0.08	12.178	12.315	0.07			**12.041**	**12.112**	0.04	1309.21
		4	12.572	12.824	0.11	12.632	12.802	0.07			**12.507**	**12.615**	0.06	1572.39
		5	12.358	12.608	0.12	12.382	12.623	0.10			**12.281**	**12.423**	0.07	1525.39
500	20	1	17.216	17.476	0.10	17.156	17.429	0.10	5	25	**16.974**	**17.129**	0.07	3718.54
		2	17.085	17.311	0.11	17.097	17.291	0.10			**16.879**	**17.052**	0.07	3762.02
		3	17.173	17.449	0.11	17.164	17.369	0.11			**16.975**	**17.148**	0.07	3849.42
		4	17.215	17.484	0.13	17.266	17.432	0.09			**16.992**	**17.166**	0.06	3687.97
		5	16.939	17.137	0.11	16.872	17.092	0.11			**16.572**	**16.786**	0.07	3693.13

Table 1: Long-term runs on Euclidean instances; results for the EAs are taken from [12].

by both EAs. For VNS the time limit was of no significance for the 100 and 250 nodes instances, whereas on graphs with 500 nodes the optimization was usually terminated due to the time constraint before 1000 successive applications of shaking without further improvement were achieved.

As there was no time information published for the EAs in [12] and since we were particularly interested in the short-term performance, we did additional experiments providing the algorithms the same, very limited amount of time. For this comparison we chose the random-key coded EA from [12] and the edge-set EA by Raidl and Julstrom [15]; the latter because it scales much better to larger instances since it derives new candidate solutions in almost linear time. For these experiments, we used instances with 500 and 1000 nodes and two different time limits for each instance size, namely 50 and 500 seconds for the 500 node graphs and 100 and 1000 seconds for the instance with 1000 nodes, respectively. For VNS k_{start} was set to 5 and $k_{\max}$ depending on the instance size. We performed 30 runs for each instance and time limit.

Table 2 lists the results. As easily can be seen VNS again performs consistently better than both EAs. The mean values of VNS with the tighter time limits are even always superior to the objective values of the overall best solutions found by both EAs with 10 times more time available. Comparing the performance of the EAs the complexity of the chromosome decoding procedure in the random-key EA becomes noticeable, and the edge-set EA always gives better results since it can perform much more iterations.

Instance			time	edge-set coded EA			random-key coded EA			VNS			
n	D	nr.	limit (sec.)	best	mean	stddev	best	mean	stddev	$k_{\max}$	best	mean	stddev
500	20	1	50	19.368	19.830	0.17	21.223	21.440	0.07	25	**17.753**	**18.108**	0.12
		2		19.156	19.522	0.13	20.836	21.097	0.09		**17.688**	**17.966**	0.10
		3		19.321	19.888	0.16	21.042	21.304	0.11		**17.799**	**18.114**	0.10
		4		19.464	19.866	0.19	21.129	21.432	0.09		**17.930**	**18.161**	0.11
		5		19.209	19.477	0.17	20.728	21.017	0.11		**17.464**	**17.863**	0.12
500	20	1	500	18.470	18.976	0.13	19.658	19.908	0.14	25	**17.290**	**17.460**	0.08
		2		18.442	18.810	0.22	19.332	19.651	0.13		**17.215**	**17.373**	0.08
		3		18.619	19.056	0.18	19.618	19.887	0.10		**17.252**	**17.464**	0.05
		4		18.745	19.116	0.17	19.654	19.905	0.11		**17.318**	**17.514**	0.07
		5		18.197	18.685	0.20	19.312	19.635	0.10		**16.932**	**17.139**	0.09
1000	25	1	100	28.721	29.265	0.16	30.996	31.288	0.11	50	**25.850**	**26.188**	0.13
		2		28.607	29.105	0.19	30.832	31.132	0.11		**25.501**	**25.981**	0.17
		3		28.410	28.905	0.17	30.515	30.856	0.12		**25.340**	**25.705**	0.09
		4		28.695	29.263	0.21	30.966	31.277	0.08		**25.562**	**26.128**	0.17
		5		28.396	28.882	0.19	30.633	31.010	0.10		**25.504**	**25.826**	0.15
1000	25	1	1000	26.494	26.936	0.14	30.097	30.401	0.13	50	**25.177**	**25.572**	0.14
		2		26.300	26.789	0.24	29.924	30.261	0.12		**25.015**	**25.342**	0.14
		3		25.762	26.556	0.21	29.586	29.981	0.12		**24.816**	**25.086**	0.11
		4		26.470	26.816	0.15	29.946	30.329	0.13		**25.289**	**25.572**	0.11
		5		26.117	26.606	0.19	29.782	30.151	0.12		**25.026**	**25.254**	0.12

Table 2: Short-term runs on Euclidean instances.

6 Conclusions and Future Work

We proposed four different types of neighborhoods for the BDMST problem, namely edge exchange, node swap, level change, and center exchange level. We combined these neighborhoods within a general VNS/VND approach and compared the results on complete Euclidean instances with those of three so-far leading meta-heuristics for this problem, namely a permutation, a random-key, and an edge-set coded evolutionary algorithm.

In both categories, solution quality as well as computation time, VNS exhibits results clearly superior to those of the EAs. In particular when the running time is strongly limited the solution quality of our VNS approach is substantially better.

In the near future we want to check the effectiveness of our VNS not only on Euclidean but also on uniform random instances, because the behavior observed when applying construction heuristics or exact approaches shows significant differences between these two classes. We also want to investigate further neighborhoods for this problem.

Another promising research direction is the hybridization of different algorithms. For example we plan to use integer linear programming approaches for the problem in order to solve larger neighborhoods, and – on the other side – we aim at boosting the performance of exact approaches by incorporating VND or VNS.

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 INFOCOM'93*, pages 1350–1358, 1993.

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

[5] 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 (IPDPS'05), workshop 12*, volume 13, page 247.1, 2005.

[6] 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.

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

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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