A Memetic Algorithm for Multi Layer Hierarchical Ring Network Design*

Christian Schauer and Günther R. Raidl

Institute of Computer Graphics and Algorithms
Vienna University of Technology, Vienna, Austria
{schauer,raidl}@ads.tuwien.ac.at

Abstract. We address the Multi Layer Hierarchical Ring Network Design Problem. This problem arises in the design of large telecommunication backbones, when high reliability is emphasized. The aim is to connect nodes that are assigned to different layers using a hierarchy of rings of bounded length. We present a memetic algorithm that focuses on clustering the nodes of each layer into disjoint subsets. A decoding procedure is then applied to each genotype for determining a ring connecting all nodes of each cluster. For local improvement we incorporate a previous variable neighborhood search. We experimentally evaluate our memetic algorithm on various graphs and show that it outperforms the sole variable neighborhood search approach in 13 out of 30 instances, while in eight instances they perform on par.

1 Introduction

In this paper we present a memetic algorithm for the *Multi Layer Hierarchical Ring Network Design* (MLHRND) problem. MLHRND arises in the field of telecommunication network design and finds applications in large, hierarchically structured networks with a strong need of survivability. It originates from a cooperation with an Austrian telecommunication provider.

Since our society increasingly depends on large and fast telecommunication networks, the matter of reliability became more and more important. In particular, it has to be avoided that larger parts of the network become disconnected in case of limited failures of devices or links. The simplest way to achieve survivability in a network is the use of a ring topology since the network stays connected in case of a single node or link failure.

For the backbone of wide area networks a single ring would not be efficient anymore. The failure of two nodes or links at the same time could disconnect large parts of the network. Moreover, requirements with respect to bandwidth and maximal delays physically limit the size of a ring. To fulfill physical constraints and ensure a high degree of survivability in larger networks, multiple interconnected rings are frequently used as backbones. To allow scalability this interconnection is often realized in a hierarchical fashion using rings on every layer of the

* This work has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT10-027.

T. Bartz-Beielstein et al. (Eds.): PPSN XIII 2014, LNCS 8672, pp. 832–841, 2014.

hierarchy. Such a network is hence called a *Hierarchical Ring Network*. If two rings are connected over a single node (*single homing*), the network can compensate for a link failure but does not stay connected if the concatenation node fails. To additionally cover this situation the rings must be connected over two different nodes on each ring, which is also called *dual homing*. The *Multi Layer Hierarchical Ring Network Design* (MLHRND) deals with a hierarchical structure spanning nodes on multiple layers using rings of bounded length and dual homing to ensure fault tolerance in case of single link and node failures. In this work we divide MLHRND into two depending subproblems, a partitioning problem to determine the nodes belonging to each ring and a ring computation problem for each partition. We propose a memetic algorithm for addressing the partitioning aspect, while a decoding procedure determines the actual rings for each partition. Furthermore, we practically evaluate this approach in experiments.

The rest of the paper is organized as follows. Related work is discussed in Section 2, while Section 3 provides a formal definition of the problem. In Section 4 we describe our memetic algorithm. Computational results are presented in Section 5. We conclude this paper in Section 6.

2 Related Work

In [8], we introduced MLHRND for the three-layer case and described a variable neighborhood search (VNS) and a greedy randomized adaptive search procedure (GRASP) for heuristically solving it. We further argued that MLHRND is NP-hard, even for the three layer case, since the classical *Capacitated Vehicle Routing Problem* can be reduced to MLHRND. To our knowledge MLHRND has not been addressed by other authors in the literature yet. However, several other problems are strongly related to it.

Balakrishnan et al. [1] describe the *Multi-Level Network Design* (MLND) problem, a generalization of the well-known Steiner network problem [4]. Nodes of the network are assigned to L different levels. For *Two-Level Network Design* ($L = 2$) the authors discuss two MIP formulations and point out that those approaches can easily be extended to a higher number of levels.

The *Ring Spur Assignment Problem* (RSAP) was introduced by Carroll and McGarraghy in [2]. The authors present a MIP formulation using connectivity constraints and an attempt for a cutting plane approach to solve larger problem instances. In RSAP the network is structured by a *tertiary ring* to which *local rings* connect to. Additionally, some nodes can be connected to a local ring by a single edge, which the authors call a *spur*. Thus, the resulting network is a three level network with rings on the top two levels and spurs on the third.

Gendreau et al. describe in [6] the *Ring Design Problem*, where nodes on a single layer are connected via interconnected rings, and propose an integer programming formulation with a quadratic objective function. The authors argue that heuristic techniques are needed to solve large size instances. Therefore, they present three ring construction heuristics based on TSP heuristics and three destroy and reconstruct approaches for post-optimization.

Proestaki and Sinclair present in [7] a variant of MLND using rings and dual homing for matters of survivability, where the node to level assignment is not given a priori but to determine during the optimization process. The objective function incorporates both the traffic on the rings and the overall ring length. As an exact approach the authors present a binary integer linear programming formulation. Additionally, they discuss a partition, construct, and perturb heuristic that iteratively, for each level, creates a solution.

Similarities further exist between MLHRND and some variations of capacitated vehicle routing problems when considering satellite depots. For instance Schwengerer et al. [9] study the *Two-Echelon Location-Routing Problem*, a combination of the VRP and the Facility Location Problem (FLP). To solve the problem the authors present a variable neighborhood search. As in MLHRND the node set is split into three subsets, which are platforms (layer 1), satellites (layer 2), and customers (layer 3). In the context of vehicle routing dual homing is not a meaningful requirement.

3 Multi Layer Hierarchical Ring Network Design

This section gives a formal definition of MLHRND and discusses some observations concerning this problem.

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. A weighting function assigns costs $c_{ij} \geq 0$ to each edge $(i, j) \in E$. Moreover, V is partitioned into $K \geq 3$ disjoint subsets $V_1, \ldots, V_K$ representing the layers each node belongs to. Edges exist between all pairs of nodes of the same and the successive layer, i.e., $E = \bigcup_{k=1,\ldots,K}(V_k \times V_k) \cup \bigcup_{k=1,\ldots,K-1}(V_k \times V_{k+1})$.

A feasible solution to MLHRND is a subgraph $G_L = (V, E_L)$ connecting all nodes in V and satisfying the following conditions; see Figure 1 for an example.

1. All nodes in V_1 are connected by a single independent ring containing no other node.
2. The remaining layers are connected by $K - 1$ respective sets of paths containing no nodes from other layers. Each node must appear in exactly one path, i.e., the paths are node and edge disjoint to ensure reliability.
3. The end nodes of each path at layer $k \in \{2, \ldots, K\}$ are further connected to two different nodes (*hubs*) in layer $k - 1$, i.e., dual homing is realized. We refer to the edges connecting paths to hubs as *uplinks*.
4. The two hub nodes, a path is connected to, must themselves be connected by a simple path at their layer, i.e., the connection to a ring may not be established via more than two layers.
5. The lengths of layer $k \in \{2, \ldots, K\}$ paths in terms of the number of edges is bounded below and above by $b_k^{\mathrm{l}} \geq 1$ and $b_k^{\mathrm{u}} \geq b_k^{\mathrm{l}}$, respectively.

The objective is to find a feasible solution with minimum total costs:

$$c(E_L) = \textstyle\sum_{(i,j)\in E_L} c_{ij}$$

From condition 1 we can conclude that finding the layer 1 ring resembles the classical Traveling Salesman Problem (TSP). Since there are no further limitations for the layer 1 ring, this subproblem can be solved independently.

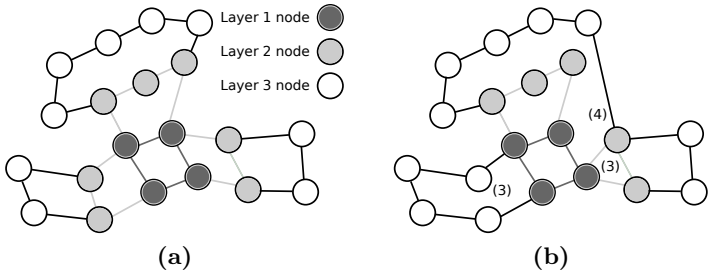

Fig. 1. Schematic representations for $K = 3$, of (a) a feasible solution and (b) an infeasible solution (the numbers in brackets indicate the violated constraints)

The situation changes when considering the remaining layers. Through the combination of the dual homing aspect (condition 3) and the connection of the hubs via simple paths (condition 4) the optimal structures of layers $k \in \{2, \ldots, K\}$ strongly depend on each other and cannot be treated separately. This combination marks the challenging aspect of MLHRND both to model and solve it. Note that relaxing either condition 3 or condition 4, each layer could be solved independently to achieve an overall optimal solution.

4 A Memetic Algorithm for MLHRND

As already mentioned, layer 1 corresponds to the classical TSP and can be solved independently. As the TSP is well studied and our primary interest lies in the structurally more complex further layers, we use in our experiments the Concorde TSP solver [3] to determine an optimal layer 1 ring. The following memetic algorithm (MA) is therefore used to solve the remaining layers. The main idea behind the MA is to split MLHRND into two subproblems: the first, to cluster the nodes of each layer into different subsets; the second, to compute Hamiltonian paths through the subsets and determine the uplinks for the paths in order to form together with the upper layer feasible rings. While the MA is used to optimize the clusters, a decoding procedure is used for calculating the paths and rings.

4.1 Representation and Decoding Procedures

A main problem, when genetic algorithms (GAs) are applied to clustering problems, is to find a proper encoding scheme for the genotype that is independent from the order of the clusters and the order of the elements within each cluster of the phenotype. An encoding scheme that compensates these problems is *linear linkage encoding* (LLE) proposed by Du et al. [5]. This representation stores for each gene the index of a fellow gene within the same cluster. By requesting that the stored index must be greater or equal than the own index, LLE provides a unique representation for each individual without the mentioned encoding problem. For MLHRND we label all nodes with indices $1, \ldots, |V|$ to define a natural

order on V. By sorting the nodes within each cluster and the clusters according to the first node of each cluster, both in ascending order, we achieve a unique representation of a candidate solution similar to LLE. Our tests showed that the computational effort for this encoding process is negligible.

Since finding a minimum weight Hamiltonian path is an NP-complete task, we implemented a heuristic decoding procedure. The idea is to first compute the path through the nodes of a cluster and in a second step determine the uplinks for the end nodes of the path. Preliminary tests showed that a decoding procedure based on the nearest neighbor principle for the TSP performed best in terms of solution quality and runtime. This procedure uses the shortest edge within a cluster as initial path, which is then iteratively extended by greedily appending nodes on both ends. The next node to be appended is always the nearest one to one of the end nodes. Ties are broken in favor of the node with the smaller index to keep the heuristic deterministic. This procedure is repeated until all nodes within a cluster are connected to a Hamiltonian path. Moreover, tests showed that improving each path with local search using a two edge exchange neighborhood structure pays off considering the increase of runtime and the improvement potential. To additionally speed up the decoding we store all decoded partitions with their resulting paths in an archive for later reuse.

Moreover, we designed a decoding strategy based on integer linear programming techniques to optimally determine the Hamiltonian path together with the uplinks. This approach allows an optimal decoding of the genotype but it performs too slowly to decode every partitioning created by the MA. Instead we use it only in the end to exactly decode the best solution found by the MA.

4.2 Initial Population

For the initial population we create one third of individuals using the randomized construction heuristic we proposed in [8]. This heuristic appends a path based on the nearest neighbor idea until b_k^{u} is reached and then starts a new path, which results in solutions with few but long paths. For this work we adapted this heuristic by also considering hub nodes and therefore to close a path in an earlier stage in case a hub node is nearer than a node on the same layer. Another third of the individuals is generated by a version of this heuristic, randomized in the same way as the first one, which creates solutions with more and shorter paths. Furthermore, we implemented another construction heuristic based on the savings principle. This heuristic iteratively creates paths for each layer by computing the savings for every pair of nodes on the layer and sorting them in descending order. Then we iterate over the savings until all nodes of the layer are connected to feasible paths, which results in solutions with many but short paths. By randomly shuffling instead of sorting the positive savings on layer 2 we obtain a randomized construction heuristic to create the remaining individuals. For the remaining layers we use the sorted savings lists, otherwise the solutions obtained would be too random. Using these three heuristics with their different solution characteristics leads to a promising and diverse initial population.

4.3 Recombination and Mutation

We designed two crossover operators using two parents and creating two descendants. Both operators recombine on the cluster level employing the same strategy for each layer $k \in \{2, \dots K\}$.

One Point Crossover (1PX) chooses a random splitting point between the list of clusters on each layer. In a first step, the clusters before the splitting point are copied from parent one to offspring one and from parent two to offspring two, respectively. Afterwards, the clusters after the splitting point are passed vice versa from parent two to offspring one and from parent one to offspring two, respectively. Now care must be taken that none of the nodes occurs twice in different clusters. Therefore, a check is needed for each node whether it can be added to the cluster or must be dropped. If too many nodes are dropped and b_k^{l} is not satisfied, the cluster is not added to the offspring at all. This means on the other hand that some nodes might be skipped. Thus, in a third step remaining nodes are iteratively added to clusters containing nodes located nearby and not fulfilling b_k^{u}. The idea is that the increase of length of the Hamiltonian path might be smaller if there are other nodes closely located to the new one. If all clusters have size b_k^{u}, then a new cluster is created and the node is added there.

Uniform Crossover (UX) decides randomly from which parent the next cluster is passed to the offspring. This means that the next cluster can either be passed from parent one or two to offspring one and correspondingly either from parent two or one to offspring two. If the number of clusters differs for the parents, the remaining clusters are added to both descendants. As in 1PX clusters not satisfying b_k^{l} are not added to the offspring at all. Again care must be taken that none of the nodes is added twice to an offspring. To add remaining nodes UX uses the same insertion strategy as 1PX.

The mutation operators are based on neighborhood structures described in Section 4.4. A mutation of an individual resembles a single random move in the respective neighborhood.

Two Node Exchange Mutation (2NE-M) swaps two nodes from different clusters on the same layer. At first, a layer, two clusters on this layer, and two nodes on the respective clusters are determined randomly. These two nodes are then swapped between the two clusters. Since no constraints could be violated this mutation is always applicable without further restrictions.

One Node Move Mutation (1NM-M) shifts a single node from one cluster to another on the same layer. The layer, the two clusters, and the shifted node are all determined randomly and then the mutation is performed. For 1NM-M care must be taken that neither b_k^{l} for the original cluster nor b_k^{u} for the destination cluster are violated. If no such clusters exist on the chosen layer then the mutation is automatically tried on another layer.

Merge Cluster Mutation (MC-M) merges two clusters on the same layer. The layer and the two clusters are chosen randomly. The two clusters must be selected so that after the merge b_k^{u} is not exceeded. In case a merge is not possible on the chosen layer an attempt on another layer is made.

Split Path Mutation (SP-M) operates on a decoded path of the phenotype and splits this path at a given position into two new paths. Since the nodes within a cluster in the genotype are stored in ascending order, dividing a cluster into two would not be meaningful. The layer and the splitting point on the path are determined randomly ensuring that the two new paths both exceed b_k^{l}. Again if a split is not possible on the chosen layer another layer is tried.

4.4 Local Improvement by Variable Neighborhood Search

For local improvement we use a downgraded version of the general variable neighborhood search (VNS) proposed in [8]. It includes an embedded variable neighborhood descent (VND), in which we consider the following neighborhood structures:

Two Edge Exchange (2EE) is applied to every path on every layer separately starting with the first uplink and ending with the second investigating all feasible candidate solutions that differ in at most two edges.

Change Uplinks (CU) finds the optimal uplinks for each path considering that the hub nodes must be different and connected via a simple path.

Split Rings (SR) divides a long path into two feasible shorter paths and connects them to the preceding layer.

Two Node Exchange (2NE) swaps two nodes from different paths on the same layer.

One Node Move (1NM) shifts a single node from one path to another on the same layer taking care that neither b_k^{l} on the original path nor b_k^{u} on the destination path is violated.

Append Rings (AR) can be seen as inversion of SR and connects the end nodes of two short paths to a feasible long path.

We did not use the *Three Edge Exchange* (3EE) and *Merge Rings* (MR) neighborhood structures from [8] because tests showed that their application was time consuming but had only a minor impact on the solution quality.

Taking a closer look one can observe that a move within 2EE and CU only affects a single path while a move in the other neighborhood structures affects greater parts of the solution. Therefore, we separated 2EE and CU from the others in an intra-VND, which is called every time a move within one of the other neighborhood structures was performed and only applied to the affected paths. The operators in the main VND are applied in the order as listed before.

For shaking we used the same strategy as in [8], i.e., $2K-2$ shaking neighborhood structures $\mathcal{N}_{1,\ldots,2K-2}$ defined as follows:

$$\mathcal{N}_i = \text{one random move in 2NE on layer } K-i+1, \quad \forall i = 1,\ldots,K-1$$
$$\mathcal{N}_i = \text{one random move in 1NM on layer } 2K-i, \quad \forall i = K,\ldots,2K-2$$

Since the VNS operates on the solution graph the shaking neighborhood structures cannot be compared with the mutation operators, which permute the clusters. Therefore, it makes sense to not only apply the VND but also to perform

shaking within the local improvement phase. In this case, we terminate the VNS if no improvement after shaking in $\mathcal{N}_{2K-2}$ was found. Therefore, the increase in runtime pays off in relation to the improvement obtained.

5 Results

For testing purposes we focus on the $K = 3$ layer scenario and use the same TSPLIB[1] based benchmark instances as in [8] for the VNS and GRASP. Additionally, we extended the test set by new random instances especially to increase the variety of smaller instances. For this purpose, we randomly placed nodes in a grid of size 10000×10000, used k-means clustering to determine the layer 1 and layer 2 nodes, and added corresponding edges. In this way we obtained a total of 74 test instances with up to 439 nodes. By using different combinations of upper bounds b_k^{u} for the path lengths we obtained 380 test cases. For the lower bound b_k^{l} we always assumed one edge as the minimum length for all paths. All these test instances can be downloaded from `www.ads.tuwien.ac.at/w/Research/Problem_Instances`.

We tested our MA using either 1PX or UX as crossover operators. As parameter setting for all test cases we used a population size of 50 individuals, tournament selection over three individuals and elitism for the best five. As stopping criterion we used a time limit as indicated in Table 1. We did not allow duplicates within the population of the same generation. The mutation rate was set to 9% for 2NE-M, 1NM-M, and MC-M and 4.5% for SP-M. We performed local improvement on the best five individuals every 500 generations. For instances with less than 300 nodes we used the VNS for larger instances the VND only, since the increase in runtime was too high. In the end, the best solution found by the MA is always improved by VNS and finally the exact decoding procedure is applied. We implemented our approach in Java 1.6 using IBM CPLEX 12.6 for the exact decoding procedure. For each of the test cases we performed 30 runs executed on a single core of an Intel Xeon (Nehalem) Quadcore CPU with 2.53GHz and 3GB of RAM. For comparison we adapted the VNS from [8] by using the intra-VND (with 3EE on the second position) with the main VND (with MR on the last position) and executed this approach on all test cases.

Results are summarized in Table 1. The columns have the following meaning: *instance* indicates the underlying graph our derived test cases are based on (with the number of nodes in the graph at the end of the name); # denotes the number of different test cases for each graph; b_2^{u} and b_3^{u} lists the different upper layer bounds, where all combinations were tested; t refers to the runtime limit in seconds; for the MA with 1PX and UX crossover and VNS the average objective values (*score*) together with their standard deviations (*dev*) are listed; columns p_{AB} show a statistical comparison between approach A and B based on a Student's t-test with an error level of 5%, $<$ ($>$) means that A performs better(worse) than B, $\approx$ indicates no significant difference between A and B.

[1] `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`

Table 1. Experimental results for the MA using either 1PX or UX as crossover operator and the VNS with average objective values and their standard deviations. Columns p_{AB} show a statistical comparison between approach A and B based on a Student's t-test with an error level of 5%. The first columns list the instance name, the number of test cases, test values for b_2^u and b_3^u, and the runtime limit for each instance.

					1PX		p_{1U} p_{1V}	UX		p_{UV}	VNS	
instance	#	b_2^u	b_3^u	t[s]	score	dev		score	dev		score	dev
ulysses22	2	4	4,6	150	123.31	0.28	$>\approx$	**123.20**	0.24	$<$	123.36	0.27
rand25	2	4	4,6	150	**71255.47**	221.34	$\approx<$	**71255.47**	221.34	$<$	72820.93	801.01
rand30	4	4	4,6	150	88484.53	3542.46	$\approx<$	**88365.69**	3422.10	$<$	89373.55	2975.65
rand35	4	4	4,6	150	89182.22	2888.34	$\approx\approx$	**89167.36**	2874.31	$\approx$	89457.32	3067.79
rand45	4	4	4,6	150	99145.13	2039.62	$\approx<$	**99126.84**	1992.77	$<$	101496.14	3640.91
att48	4	4	4,6	150	59663.81	632.16	$\approx<$	**59596.17**	681.46	$<$	60169.25	966.66
eil51	4	4	4,6	150	743.72	15.56	$\approx\approx$	**743.02**	15.63	$\approx$	746.89	16.28
berlin52	4	4	4,6	150	13370.03	201.04	$\approx<$	**13364.96**	197.50	$<$	13451.39	279.67
rand55	4	4	4,6	150	**111846.88**	2971.71	$\approx<$	111882.44	2878.37	$<$	113360.76	2299.75
rand70	12	4,7	4,7,11	150	126402.60	2322.23	$><$	**125926.40**	2264.46	$<$	127383.39	3835.20
eil76	12	4,7	4,7,11	150	902.71	21.82	$\approx\approx$	**902.03**	23.09	$\approx$	903.94	24.27
rand85	18	4,7	4,7,11	150	136312.04	3237.14	$\approx<$	**136050.13**	3227.32	$<$	136995.32	3755.33
gr96	18	4,7	4,7,11	300	**925.80**	21.40	$\approx<$	926.07	22.21	$<$	930.99	21.11
kroA100	18	4,7	4,7,11	300	40043.51	1132.73	$\approx\approx$	**39992.35**	1151.43	$\approx$	40066.80	1152.69
kroB100	18	4,7	4,7,11	300	**40759.91**	1068.90	$\approx\approx$	40780.24	1122.37	$\approx$	40826.96	1219.67
bier127	12	7,11	7,11,14	300	202817.43	2601.11	$\approx>$	202674.00	2836.98	$>$	**201500.32**	2445.40
ch150	18	7,11	7,11,14	300	11719.20	236.62	$\approx>$	11715.27	246.43	$>$	**11638.03**	230.24
rand175	18	7,11	7,11,14	300	184434.64	4073.46	$\approx>$	184749.36	4161.79	$>$	**182736.31**	3843.28
kroA200	18	7,11	7,11,14	300	53045.74	986.66	$<>$	53183.41	1089.88	$>$	**52854.56**	1018.72
kroB200	18	7,11	7,11,14	300	53148.67	1024.49	$\approx>$	53229.85	1023.96	$>$	**52984.27**	1020.04
gr229	12	11,16	11,16,19	600	2840.61	66.71	$\approx>$	2842.48	65.43	$>$	**2821.13**	71.97
rand250	12	11,16	11,16,19	600	214841.28	2983.39	$\approx>$	215245.46	3347.49	$>$	**212409.05**	2223.95
rand275	18	11,16	11,16,19	600	219517.82	4488.71	$<>$	220505.76	4672.04	$>$	**217118.47**	3608.25
pr299	18	11,16	11,16,19	600	88667.95	1264.57	$<>$	88920.18	1383.50	$>$	**87873.12**	1033.80
lin318	18	11,16	11,16,19	600	77173.81	1718.17	$<\approx$	77612.38	1835.81	$>$	**77019.39**	1264.34
rand350	18	11,16	11,16,19	600	**248766.28**	5152.62	$<<$	250009.87	5706.74	$\approx$	250113.76	4810.65
rand375	18	11,16	11,16,19	600	**259471.79**	5313.26	$<\approx$	261068.47	6000.87	$>$	259992.72	5100.66
rand400	18	11,16	11,16,19	900	269699.91	5854.83	$<\approx$	271924.26	6542.30	$>$	**269349.24**	5199.97
gr431	18	11,16	11,16,19	900	**3373.65**	57.92	$<<$	3401.35	63.94	$<$	3455.16	65.91
pr439	18	11,16	11,16,19	900	**201790.51**	6992.98	$<<$	204776.11	7414.42	$<$	206108.42	8830.63

Firstly, we observe that by using our new VND approach we are able to improve the VNS results significantly compared to [8]. The GA without local improvement cannot compete with the VNS except for rand25. Therefore and due to space limitations, we do not present the results for the GA alone. Of greater interest is the comparison between the MA and VNS. For all 15 instances with up to 100 nodes the MA with both crossover operators outperforms the VNS in terms of solution quality, in ten cases the difference is significant (see columns p_{AB}). In many cases UX delivers the best results. For instances of this size the MA can cover a large search space, while the VNS gets stuck in a local optimum too early. The situation changes for instances with 127–300 nodes, were VNS always performs significantly better. Here the VNS can show its strength and cover a larger search space because the application of VND is still fast and many iterations are performed. For more nodes the MA, especially with 1PX, performs again better than the VNS except for lin318 and rand400, where the VNS is not significantly better, and for rand375, where 1PX is not significantly better.

As already mentioned the application of VND becomes very time consuming for larger instances and only some iterations for the VNS are possible in the given time limit. The MA still can compute at least 3000 generations especially with 1PX, which in practice is faster than UX. Therefore, we conclude that especially for larger instances the MA is the algorithm of choice.

6 Conclusions and Future Work

We presented a memetic algorithm to solve the Multi Layer Hierarchical Ring Network Design problem. The basic concept is to use the MA to cluster the nodes of each layer into disjoint subsets, while a decoding procedure computes a Hamiltonian path through each cluster and finds uplinks to determine a feasible solution. The approach includes a variant of a previously published VNS with an embedded VND. The VND has been adapted by distinguishing between neighborhood structures that work on the path level or on the whole solution. In this way we were able to speedup the VND significantly and also obtained better results for the VNS alone. The MA outperforms the VNS on instances with up to 100 and more than 350 nodes. In the future we will focus on decomposition techniques, e.g., Benders decomposition for solving medium sized instances exactly, and the design of large neighborhood structures.

References

1. Balakrishnan, A., Magnanti, T.L., Mirchandani, P.: The Multi-level Network Design Problem. Tech. Rep. 3366-91, Massachusetts Institute of Technology (1991)
2. Carroll, P., McGarraghy, S.: Investigation of the ring spur assignment problem. In: Bigi, G., Frangioni, A., Scutellà, M. (eds.) Proceedings of the 4th International Network Optimization Conference (INOC 2009). pp. MB1–3 (2009)
3. Cook, W.J.: Concorde TSP Solver, `http://www.math.uwaterloo.ca/tsp/concorde/` (accessed: March 13, 2014)
4. Dreyfus, S., Wagner, R.A.: The Steiner Problem in Graphs. Networks 1, 195–207 (1972)
5. Du, J., Korkmaz, E., Alhajj, R., Barker, K.: Novel clustering approach that employs genetic algorithm with new representation scheme and multiple objectives. In: Kambayashi, Y., Mohania, M., Wöß, W. (eds.) DaWaK 2004. LNCS, vol. 3181, pp. 219–228. Springer, Heidelberg (2004)
6. Gendreau, M., Labbé, M., Laporte, G.: Efficient heuristics for the design of ring networks. Telecommunication Systems 4(1), 177–188 (1995)
7. Proestaki, A., Sinclair, M.: Design and dimensioning of dual-homing hierarchical multi-ring networks. IEE Proceedings Communications 147(2), 96–104 (2000)
8. Schauer, C., Raidl, G.R.: Variable Neighborhood Search and GRASP for Three-Layer Hierarchical Ring Network Design. In: Coello Coello, C.A., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012, Part I. LNCS, vol. 7491, pp. 458–467. Springer, Heidelberg (2012)
9. Schwengerer, M., Pirkwieser, S., Raidl, G.R.: A variable neighborhood search approach for the two-echelon location-routing problem. In: Hao, J.-K., Middendorf, M. (eds.) EvoCOP 2012. LNCS, vol. 7245, pp. 13–24. Springer, Heidelberg (2012)