JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 26, 1231-1242 (2010)

Oriented Networks Design Problem

JEROME GALTIER[1], IGOR PESEK[2], KATJA PRNAVER[3]
AND JANEZ ŽEROVNIK[3,4]
[1]*Orange Labs 905 rue Albert Einstein*
06921 Sophia Antipolis Cedex, France
[2]*Faculty of Natural Sciences and Mathematics*
University of Maribor
Koroška 160,2000 Maribor, Slovenia
[3] *Institute of Mathematics, Physics and Mechanics*
Jadranska 19, 1000 Ljubljana, Slovenia
[4]*Faculty of Mechanical Engineering*
University of Ljubljana
Aškerčeva 6, 1000 Ljubljana, Slovenia

The design of interconnection networks is very popular in computer science. In this paper we introduce a novel problem, the design of oriented networks. The underlying structure of such networks is a directed graph with weights on the arcs, which are the number of paths of the routing that traverse the arc. The cost of a network with a routing is defined as a sum of arc costs that are computed with concave increasing cost function depending on weights over directed arcs. The objective is to find the routing that is optimal in terms of costs. The corresponding minimization problem is approached with a local search type heuristics. New local search neighborhood is defined and analyzed.

Keywords: oriented networks problem, heuristics, local search, hill climbing, SDH networks

1. INTRODUCTION

One of the most popular applications of graph theory in computer science is the design of interconnection networks. A desirable network is usually optimal with respect to the value of one (or more) graph invariants, for example it has small diameter, small average distance and small routing costs or allows uniform distribution of traffic. In this paper we look closely at the oriented network design problem [1] that arises in the design of SDH loop networks [4, 5]. Synchronous Digital Hierarchy (SDH) is a multiplexing protocol for transferring multiple digital bit streams using lasers or light-emitting diodes (LEDs) over the same optical fiber. SDH standard was developed by the International Telecommunication Union (ITU), and is documented in standard G.707 and its extension G.708. In such networks, the traffic requests are routed along a set of containers of fixed capacity, where each container connects two nodes of the network. The cost of the network is defined as the sum of arc contributions, where the arc contribution is a certain power ($w(e)^{\alpha}$, $0 \le \alpha \le 1$) of the number of paths (of the routing) traversing the arc. Practitioners discovered that using a concave increasing cost function of the form $w(e)^{\alpha}$ accurately describes the problem. Consequently, the cost depends only on the capacity and no cost of

Received July 11, 2008; revised October 20, 2008 & January 5 & May 7 & June 19, 2009; accepted July 10, 2009.
Communicated by Nancy M. Amato.

introducing an arc is present. While the problem given in [1] asks simply for optimal networks, it may also be of interest to find extremal cases for given number of nodes and arcs. In this paper we define a corresponding optimization problem that may be attacked with any metaheuristics for discrete optimization. We define a neighborhood that enables us to run any local search based algorithm and prove some properties of the neighborhood.

2. THE PROBLEM

A network is a directed arc weighted graph $G(V, A, \Lambda)$, where V is the set of vertices A is the set of directed edges or arcs and Λ gives integral weights to the arcs. The weight of an arc can be understood as the amount of traffic that is allowed along the arc. In an optimal network the amount of allowed traffic on an arc is always equal to the number of times the walks of the optimal complete routing traverse it.

A (*complete*) *routing* R is a collection of walks, one for each ordered pair of distinct vertices. Formally, $R = \{p_{u,v} | u, v \in V\}$ where $p_{u,v}$ is a walk from u to v. For a request of one unit routed through $p_{u,v}$, the walk adds one unit to the weight of each arc that it traverses. In general, a walk may traverse the same arc more than once contributing each time to the weight. It is clear that in any solution that minimizes a function of the total weight all walks must be paths, *i.e.* walks that meet each vertex at most once and consequently traverse each arc at most once.

If the requests are all equal to one unit, the weight $\Lambda(i, j)$ of an arc (i, j) is the number of times the walks of the routing traverse it. The cost of the arc is then defined as $\Lambda(i, j)^{\alpha}$ and the cost of the routing R is computed as the sum of the arc costs, $COST(R) = \sum \Lambda(i, j)^{\alpha}$.

The restricted problem can be formulated as a minimization problem as follows:

Input: Given are α, $0 \le \alpha \le 1$ and the number of vertices n.
Task: Find a (complete) routing where the corresponding cost is minimal.

Once we have the routing with minimal costs, the optimal network can clearly be obtained by simply setting container capacities with the weights of the routing.

A more general problem, where each ordered pair of vertices has a communication weight $w(u, v)$, can be studied by simply defining the weight of an arc to be the sum of weights of the pair that uses it for communication. More formally,

$$COST(R) = \sum \Lambda(i, j)^{\alpha}, \text{ where } \Lambda(i, j) = \sum_{i,j \in p\, u,\, v \in R} w(w, v).$$

Note that for $0 \le \alpha \le 1$, an optimal routing of requests will never be split. Indeed consider that two paths are used to route the same request. Then the strategy of carrying the entire request on one of the paths (the best one) is necessarily improving the solution.

Note also that the case $\alpha = 1$ is trivial, since the best strategy is then to route all the requests through paths of length one.

2.1 Exact Solutions

We have exhaustively compared all the possible network configurations for small

number of vertices. The results are as follows.

For 3 nodes there are only two possible graphs for optimal solutions. The first, which is optimal for $\alpha \in \left[0;\ \frac{\ln 2}{\ln 3}\right]$ is the oriented ring. The second, optimal for $\alpha \in \left[\frac{\ln 2}{\ln 3},\ 1\right]$ is the complete graph.

For 4 nodes, things are slightly more complicated. The two extreme optimal configurations remain the oriented ring and the complete graph. In between, however, a particular combination shown in Fig. 1 occurs. The first change of optimal configuration occurs when $\alpha \approx 0.5807$ and the second change takes place when $\alpha \approx 0.6437$.

Optimal solutions for graphs $n \geq 5$ were not computed since search space is too large for exhaustive search, which indicates that the problem stated may be extremely difficult. Note that the general problem we are studying is a strongly NP-complete problem, and we try to get good solutions in a restricted case. Therefore the use of metaheuristics is natural and here we started with local search based metaheuristics.

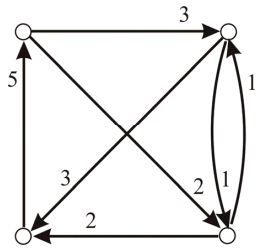


Fig. 1. Optimum for 4 nodes and $\alpha \in [0.5807, 0.6437]$.

3. LOCAL SEARCH

In order to apply the local search paradigm, we need to define the search space, the cost function, the selection rule for the initial solution and the neighborhood structure.

3.1 Initial Solution

Since all possible configurations can be reached with our local search from two extremal configurations of the network (for proof see section 3.5) we use complete graph and a cycle on all vertices (we will refer to it as *oriented ring*) as our initial configurations. We will later discuss some generalizations where any network can be used for the initial solution of the local search.

3.2 Cost Function

In order to determine the cost of a particular network, we use the cost function which was defined in section 2,

$$COST(R) = \sum_{(i,j)\in A(R)} \Lambda(i,j)^{\alpha}$$

where R is a routing of the network. However, there are many other cost functions that may be of interest. In particular, it is more accurate for developing fiber networks to include some base cost of a link, for example with

$$COST(i, j) = BASE + \Lambda(i, j)^{\alpha}.$$

Furthermore, in a more general setting, the cost function can be any concave increasing (sublinear) function (see [1] page 4). In principle, the local search heuristic with the neighborhood defined below can be applied to any cost function. The analysis of its behavior may however be much more complicated.

3.3 Search Space

We define the search space to be the set of all (complete) routings, thus allowing any walks. We take this decision for simplicity, so that it is never necessary to check if the walk is not simple. However, one can always repair the final solution easily by replacing each nonsimple walk by a path and thus decrease the overall configuration cost.

3.4 Neighborhood

In our definition of the neighborhood, all neighbors are obtained as follows. We select three vertices i, j and k with additional condition that $\Lambda(i, k) > 0$. Then we subtract L paths from $\Lambda(i, k)$ and add L paths to $\Lambda(i, j)$ and $\Lambda(i, k)$. For graphic representation of the change see Fig. 2 where we have set $L = 1$. More precisely, we reroute L paths from arc (i, k) to arcs (i, j) and (j, k). Since the described neighborhood looks like a triangle, we call it *Triangle neighborhood*. It is important to note that we always maintain the complete graph in the background, making it possible to add some load to a non-existing arc in the current configuration and therefore to extend the search space. One could naturally define other neighborhoods that would move load from arc(s) to arcs on a square or on a larger cycle (with suitable orientations). However, we will prove that the triangle neighborhood is surprisingly powerful. In next section we will prove that each optimal network can be reached from the complete graph with triangle neighborhood moves.

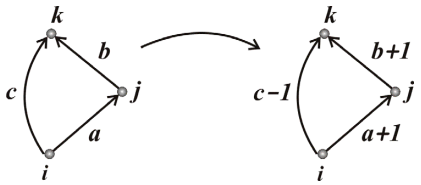


Fig. 2. Triangle neighbourhood.

In order to be able to focus on the triangle neighborhood only, we will show that the cost of rerouting L paths can be computed only by weights of involved arcs. For this purpose, we define *GAIN* of routing as follows. Denote by R_1 the graph before move and by Λ_1 the weights on arcs before move, and by R_2 the graph after move with corresponding weights Λ_2. Then

$$COST(R_1) = \sum_{a,b \in E(R_1)} (\Lambda_1(a,b))^\alpha$$
$$= (\Lambda_1(i,j))^\alpha + (\Lambda_1(i,k))^\alpha + (\Lambda_1(j,k))^\alpha + \sum_{a,b \in E(R_1 \setminus \{(i,j),(i,k),(j,k)\})} (\Lambda_1(a,b))^\alpha,$$
$$COST(R_2) = \sum_{a,b \in E(R_2)} (\Lambda_2(a,b))^\alpha$$
$$= (\Lambda_2(i,j))^\alpha + (\Lambda_2(i,k))^\alpha + (\Lambda_2(j,k))^\alpha + \sum_{a,b \in E(R_1 \setminus \{(i,j),(i,k),(j,k)\})} (\Lambda_1(a,b))^\alpha$$
$$= (\Lambda_1(i,k) - L)^\alpha + (\Lambda_1(j,k) + L)^\alpha + (\Lambda_1(i,j) + L)^\alpha + \sum_{a,b \in E(R_1 \setminus \{(i,j),(i,k),(j,k)\})} (\Lambda_1(a,b))^\alpha,$$
$$GAIN(R) = COST(R_1) - COST(R_2) = (\Lambda_1(i,j))^\alpha + (\Lambda_1(i,k))^\alpha + (\Lambda_1(j,k))^\alpha$$
$$- (\Lambda_1(i,k) - L)^\alpha - (\Lambda_1(j,k) + L)^\alpha - (\Lambda_1(i,j) + L)^\alpha.$$

On the other hand, let us observe the local change only and define it as the function *DELTA*(a, b, c) as the profit of moving L paths from the arc with weight c to arcs that have weights a and b. For the move depicted on Fig. 2 where $L = 1$ we have

$$DELTA(a, b, c) = a^\alpha + b^\alpha + c^\alpha - ((a + 1)^\alpha + (b + 1)^\alpha + (c - 1)^\alpha).$$

In general, when moving L paths

$$DELTA(a, b, c) = a^\alpha + b^\alpha + c^\alpha - ((a + L)^\alpha + (b + L)^\alpha + (c - L)^\alpha).$$

Furthermore, from the definition it follows $\Lambda_1(i, j) = a$, $\Lambda_1(j, k) = b$ and $\Lambda_1(i, k) = c$, and hence

$$GAIN(R) = DELTA(\Lambda(i, j), \Lambda(j, k), \Lambda(i, k)).$$

3.5 Properties of the Triangle Neighbourhood

Theorem 1 Let $G = (V, E)$ be a digraph with edge e having weight $w(e)$ admitting an all-to-all routing with optimal cost. It is possible to obtain G from the complete graph by local triangular transformations.

Proof: Since the optimal graph for $\alpha = 1$ is the complete graph, no other transformation is required. Otherwise, for $\alpha < 1$, for each pair (u, v) of distinct vertices, let us associate the path from u to v in the optimal routing for G,

$$p_{u,v} = x_0, x_1, \ldots, x_{l(u, v)}$$

where $l(u, v)$ is the length of $p_{u,v}$, x_0 is u, $x_{l(u,v)}$ is v and $(x_i, x_{i+1}) \in E$ for $i = 0, \ldots, l(u, v) - 1$.

This path has the property that, if $0 \le i < j \le l(u, v)$, then the path from x_i to x_j in the optimal routing is a subpath of $p_{u,v}$: $p_{x_i x_j} = x_i, x_{i+1}, \ldots, x_j$ and $l(x_i, x_j) = j - i$. Indeed if it is not the case, since $\alpha \in (0, 1)$, one of the following operations will give a strictly better network than G:

- replace $p_{u,v}$ by $x_0, \ldots, x_{i-1}, p_{x_i,x_j}, x_{j+1}, \ldots, x_{l(u,v)}$,
- replace p_{x_i,x_j} by $x_i, \ldots, x_j$.

Let a demand consist of a source, a destination, a path and a weight. We proceed by induction on sets of demands denoted by D^k. D^0 is the set of all pairs (u, v) with path $p_{u,v}$ and weight 1. Suppose D^k contains at least one demand (u, v) with $l(u, v) \geq 2$ and has the property that for any demand from D^k it also contains all demands (s, t) with $l(s, t) < l(u, v)$. Then select a demand (u, v) in D^k with longest path length $l(u, v)$ and one vertex x_i in $p_{u,v}$ distinct from u and v (that is $0 < i < l(u, v)$). Then $l(u, x_i) = i < l(u, v)$ and $l(x_i, v) = l(u, v) - i < l(u, v)$. So, both (u, x_i) and (x_i, v) are in D^k. Construct D^{k+1} from D^k by suppressing (u, v) and increasing weights of (u, x_i) and (x_i, v) by the weight of (u, v). In other words, we delete one of the demands with longest length and add its weight to two demands with shorter length. Note that D^{k+1} contains all demands (s, t) with $l(s, t) < l(u, v)$. Hence the new set of demands has less demands and the property assumed above is preserved.

The process is continued until D^K contains only demands of length 1. Looking at the graphs of source/destination pairs, we notice that D^0 corresponds to the complete graph, and D^K to G. Moreover, for all $k \in \{0, \ldots, K - 1\}$, the graph of D^{k+1} is obtained from the graph of D^k by a local triangular transformation, and the corresponding weights are exactly copied by the operation. ❑

3.6 Cleaning the Solution

For the implementation of the triangle neighborhood there is no need to know the paths that traverse arcs. During our experiments we found examples in which the weights were too large because some "paths" became walks with cycles. We have therefore implemented a straightforward procedure which finds possible cycles that may arise after the last move, and decreases the weights accordingly. We omit the details.

3.7 Local Search Metaheuristics

The triangle neighborhood, for all values $n \geq 5$ that we consider, is a large neighbourhood and its exhaustive exploration turned out to be rather impractical. Therefore for this neighborhood, we resort to a simple hill-climbing strategy, based on a randomized selection combined with threshold accepting [2]. More precisely, at each iteration we draw a random move and compute its cost. The move is accepted, either if the GAIN of the current move is positive, hence it is improving the solution, or if the GAIN is greater than some fixed threshold value, hence the cost of the solution is worsened slightly. Otherwise the state remains unchanged. The procedure stops after a fixed number of iterations.

Additionally we added a *kick*, which is used after m nonimproving iterations in our search and is used as escape move from local minima. In each kick we use a second neighborhood that employs the same idea as described in the triangle neighborhood with one important difference. Here we randomly distribute stepwise entire load (not only L) from the selected arc to other arcs. One kick consists of three consecutive moves using the second neighborhood.

4. EXPERIMENTAL ANALYSIS

In order to determine successfulness of our new approach, we first explain our experimental settings, then describe post processing steps, present the experimental results, and finally give a formula which describes successfulness of the described schema.

4.1 Experimental Setting

Experiments were performed on an Intel Xeon (2.0 GHz) processor running Windows Server 2003. The algorithms have been coded in C++ exploiting the framework EasyLocal++ [3], and the executables were obtained using the GNU C/C++ compiler (ver. 3.4.1).

The stopping criterion in our experiments is based on the number of iterations, *i.e.* the number of the network configurations examined and was set to 9000 iterations.

Additionally we have set the number of nonimproving iterations $m = 50$ for the *kick*, $L = 1$ and we have used a complete graph and a ring for our initial configuration.

We run a search for optimal configuration for every α in the range $0 \le \alpha \le 1$ with the variable precision set to 0.001, *i.e.* for $\alpha = 0, 0.001, 0.002, 0.003, \ldots, 0.999, 1$.

4.2 Post Processing the Results

After a number of independent local search runs, the results of the experiment were filtered with a post processing procedure which is independent to local search presented above. In our experiments we did five runs for each α, each run with different random seed. Since our local search method does not always find the optimal solution but only some local minima, we needed a method to filter out only the best configurations. Additionally we look at the independent runs as one multistart run of our heuristics. From the properties of the problem we also intuitively expect that best configurations apply to some range of α.

Let us now describe the post processing step. First we scan all the results from $0 \le \alpha \le 1$ in the ascending order of α with variable precision set to 0.001. For each α, we determine the best configuration and then compare the cost of current best configuration against the best configurations that were found previously. If some configuration is better than the current best in terms of cost, then the current configuration is discarded in favor of the better configuration. When we finish scanning in ascending order we repeat the procedure in descending order. With this method we greatly improve the results summary since we remove some configurations that may be very good but are not the best.

4.3 Experimental Results

In this section, we present the results for graph with sizes $n = 5$, 6, 7 and 20. We also compare our results against the results obtained with the conjecture found in [1], which states that the optimal configurations are the star like graphs.

For $n = 5$ we found four (4) best configurations for entire interval of α. It is interesting to observe that the second configuration is optimal only on a very short interval for α. Furthermore, our heuristic finds all the optimal configurations on same intervals as

predicted by the conjecture. Results are given in Table 1, where optimal solutions are in bold typeface and corresponding graphs are shown on Figs. 3 to 6.

Table 1. Best configurations for $n = 5$.

α	Our local search	Conjecture
$\alpha \in [0, 0.512]$	oriented ring (Fig. 3)	oriented ring
$\alpha \in [0.513, 0.514]$	2 × cycles (Fig. 4)	2 × cycles
$\alpha \in [0.515, 0.661]$	4 × cycles (Fig. 5)	5 × cycles
$\alpha \in [0.662, 1]$	complete graph (Fig. 6)	complete graph

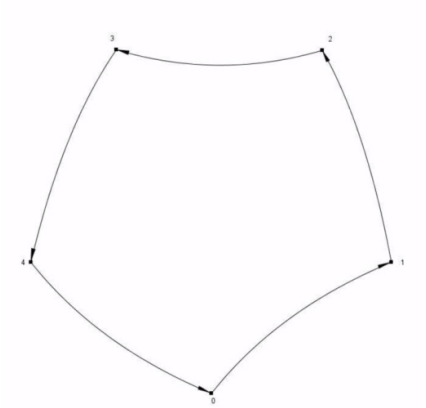


Fig. 3. Oriented ring on 5 nodes, optimal for $\alpha \in [0, 0.512]$.

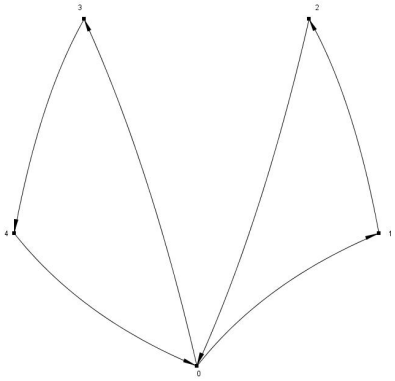


Fig. 4. Graph with 2 × 3-*cycles* on 5 nodes, optimal for $\alpha \in [0.513, 0.514]$.

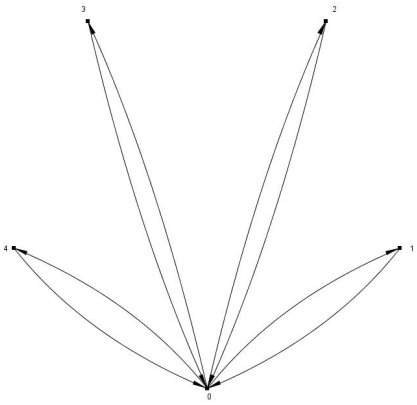


Fig. 5. Graph with 4 × 2-*cycles* on 5 nodes, optimal for $\alpha \in [0.515, 0.661]$.

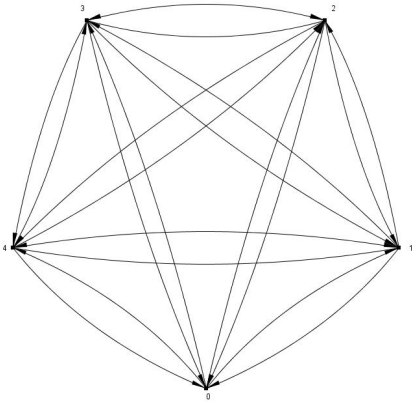


Fig. 6. Complete graph on 5 nodes, optimal for $\alpha \in [0.662, 1]$.

For $n = 6$ we found four (4) best configurations for given interval of α and are presented in Table 2. Again the results of heuristics are consistent with the conjecture.

Table 2. Best configurations for $n = 6$.

α	Our local search	Conjecture
$\alpha \in [0, 0.446]$	oriented ring	oriented ring
$\alpha \in [0.447, 0.488]$	1×2-cycles and 2×3-cycles	1×2-cycles and 2×3-cycles
$\alpha \in [0.489, 0.682]$	5×2-cycles	5×2-cycles
$\alpha \in [0.683, 1]$	complete graph	complete graph

Table 3. Best configurations for $n = 7$.

α	Our local search	Conjecture
$\alpha \in [0, 0.388]$	oriented ring	oriented ring
$\alpha \in [0.389, 0.430]$	oriented ring	3×3-cycles
$\alpha \in [0.431, 0.474]$	6×2-cylces	3×3-cycles
$\alpha \in [0.475, 0.699]$	6×2-cycles	6×2-cycles
$\alpha \in [0.7, 1]$	complete graph	complete graph

Finally, for $n = 7$ we found four (4) best configurations which are presented in Table 3. For the first time, we see a difference between the two methods. On one (small) interval the best result of the heuristics is the oriented ring which is worse than the star graph consisting of two 3-cycles. On all the other intervals the results of the heuristic search and the conjecture are consistent.

As expected, some configurations dominate over large intervals of α. It is important to note, that even if we increase n, the computational times for our local search approach does not increase significantly. This is because we randomly choose one candidate at each iteration and check its cost, which implies that the increase in computational time is only due to the computation of the cost for some configuration.

For larger networks it is much more difficult to asses how good particular scheme is. Therefore we define a formula involving straightforward lower and upper bounds for the optimal solution

$$(n-1)^{1+\alpha} \le Optimal(n) \le \min\left\{2(n-1)^{1+\alpha},\ \left(\frac{n(n-1)}{2}\right)^{\alpha},\ n(n-1)\right\}$$

where $2(n-1)^{1+\alpha}$, $\left(\frac{n(n-1)}{2}\right)^{\alpha}$, $n(n-1)$ correspond to the cost of star topology, oriented ring and complete oriented topology, respectively. Clearly, using this formula we can quickly determine how successful proposed scheme is.

We did a final run on 20 nodes ($n = 20$) and compared the results against the above bounds. For most values of α the results of heuristics are equal to the upper bound. This is not surprising because the solutions found are oriented ring, star graphs or complete graphs that are conjectured to be optimal at certain intervals. However, the heuristics also found a network which is on interval $\alpha \in [0.21, 0.37]$ better than the scheme described

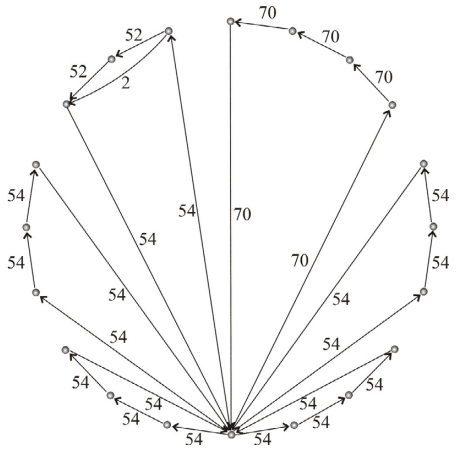


Fig. 7. Optimal network for 20 nodes on interval $\alpha \in [0.21, 0.37]$.

above. The network found is depicted on Fig. 7. We can conclude that the conjecture of [1] does not provide all optimal configurations and also that the proposed heuristics is capable of finding new near optimal or optimal solutions.

5. CONCLUSIONS AND FUTURE WORK

In our implementation a complete run (1000 values for α, 9000 iterations for each run, and post processing) takes approximately 6.2, 7.1, 8 and 22 hours CPU for n = 5, 6, 7 and 20, respectively. The results for larger n are approximately consistent with the conjecture [1]: most (but not all) optimal configurations are found and for each α the solution found by the local search is at most 4% worse that the conjectured configuration. Hence on the one hand the experiment supports the conjecture [1], and, on the other hand the small differences give some evidence that the local search used here gives good near optimal solutions.

Note however that the local search with the special neighborhood defined here can be applied to a more general problem with both variable communication demands between pairs of nodes and additional cost for initiating an arc. In practical applications this generalization presumably models the optimization problem much better and hopefully the method proposed here would probably give competitive solutions.

Furthermore, once we have the basic local search, it is natural and possibly rewarding to use more sophisticated heuristics that could further improve the solutions. We plan to develop such heuristics and test it in same experimental setting to determine the efficiency of the proposed approach.

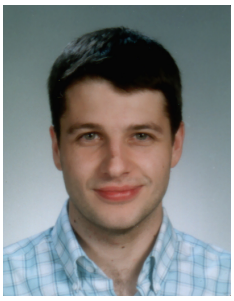
ACKNOWLEDGMENT

The authors would like to thank to the referees for their valuable comments on the original manuscript which helped to significantly improve the presentation and clarity of the paper. Authors were partially supported by Slovenian research agency ARRS.

REFERENCES

1. S. Choplin, J. Galtier, and S. Perennes, "Optimal concave costs in the SDH context," Technical Report, INRIA Rapport de recherche No. 5201, 2004.
2. G. Dueck and T. Scheuer, "Threshold accepting, a general purpose optimization algorithm appearing superior to simulated annealing," *Journal of Computational Physics*, Vol. 90, 1990, pp. 161-175.
3. L. D. Gaspero and A. Schaerf, "EasyLocal++: An object-oriented frame-work for flexible design of local search algorithms," *Software – Practice and Experience*, Vol. 33, 2003, pp. 733-765.
4. H. G. Perros, *Connection-Oriented Networks: SONET/SDH*, *ATM*, *MPLS and Optical Networks*, John Wiley and Sons, Chichester, 2005.
5. B. Sanso and P. Soriano, eds., *Telecommunications Network Planning*, Springer, Berlin, 1999.

Jerome Galtier is a former student of the Ecole Normale Superieure of Paris and received his Ph.D. from the University of Versailles Saint-Quentin, France, in 1997. He also graduated form the Sup'Telecom Paris Tech institute as an engineer the same year. He works now with France Telecom and wrote his habilitation thesis in 2008. His area of research concerns algorithmics and operation research for the telecommunication industry.



Igor Pesek recieved his B.Sc. from Faculty of Education in 2002 and his Ph.D. from Faculty of Electrotehnics, Computers and Informatics, University of Maribor, Slovenia, in 2009. He works now at the Faculty of Natural Sciences and Mathematics, Maribor, Slovenia. His main research interest include oriented network design problem, various scheduling problems and e-learning authoring tools.

Katja Prnaver started her University study in 2000 at the Faculty of Electrical Engineering and Computer Science at University of Maribor. In 2002, she started a parallel study of Mathematics at the Faculty of Education (now Faculty of Natural Sciences and Mathematics). In 2005, she received B.Sc. degree in Computer Science. In 2006, she has been offered a position as Young researcher at the institute of Mathematics, Physics and Mechanics, Ljubljana, Slovenia, where she is still employed. She

started her Ph.D. study of Mathematics in the same year 2006. She also completed the undergraduate study of Mathematics in 2007, to receive her second B.Sc. diploma. Her research so far has been done on Total Coloring of graphs and oriented networks; currently she is doing extensive research on Retracts and Graph bundles and is going to do 6 months research project in the field of Warehouse management at Danish Technical University in August 2009.

Janez Žerovnik received the B.S. and M.S. degree in Mathematics from University of Ljubljana, Slovenia in 1982. In 1992 he received Ph.D. degree in Computer Science from University of Ljubljana and in 1994 Ph.D. degree in Mathematics from Technical University Graz, Austria. Currently, he is Professor of Mathematics at the Faculty of Mechanical Engineering at University of Ljubljana and part time researcher at the Institute of Mathematics, Physics and Mechanics. His research interests include discrete mathematics, in particular graph theory and its applications in computer science, operational research, mathematical chemistry, *etc*.