Paper

Heuristic algorithms in topological design of telecommunication networks

Piotr Karaś

Abstract — The paper addresses the generic topological network design problem and considers the use of various heuristic algorithms for solving the problem. The target of the optimisation is to determine a network structure and demand allocation pattern that would minimise the cost of the network, which is given by fixed installation costs of nodes and links and variable link capacity costs described by linear or concave functions. Input data for the optimisation consists of a list of potential node and link locations and their costs and a set of demands defined between the nodes. Since the problem is known to be NP-hard, the use of specialised heuristic algorithms is proposed. The presented approaches encompass original ideas as well as selected methods described in literature and their enhancements. The algorithms are based on the following ideas and methods: shifting of individual flows, local and global restoration of flows from chosen links or nodes, Yaged algorithm for finding local minima, Minoux greedy algorithm, simulated allocation and genetic algorithms. Efficiency of each of the proposed methods is tested on a set of numerical examples.

Keywords — topological design, network optimisation, heuristic algorithms, genetic algorithms.

1. Introduction

Topological design of telecommunication networks encompasses a range of problems related to localisation of links and nodes of a network. The target of the optimisation is to determine a network structure and demand allocation pattern that would minimise the cost of the network, given a list of potential node locations and a list of admissible interconnections between these nodes. The objective function to be minimised is given as a sum of fixed installation costs of nodes and links and variable link capacity costs (a function of link capacity).

Two subproblems can be distinguished – namely the link localisation problem (LLP), where only link localisation is to be optimised, and the more general transit node and link localisation problem (TNLLP), where localisation of links and transit nodes is subject to optimisation. In literature the LLP is also referred to as the optimal network design problem. This paper addresses both variants, however it concentrates on the more complex TNLLP.

Below the inputs and objectives of the optimisation are described in more detail.

Input data for the optimisation:

- a list of nodes, where access nodes (which originate or terminate demands) and transit nodes (which may transit flows) are distinguished;
- a list of all allowable interconnections between nodes (access links for connecting access nodes to transit nodes and transit links for interconnecting transit nodes);
- demands defined between access nodes, which are to be satisfied;
- link costs – a fixed link installation fee and a variable cost (function of link capacity);
- node costs – a fixed transit node installation fee.

Objectives of the optimisation:

- derive a set of necessary transit nodes;
- derive a set of necessary transit links and access links;
- find an optimal routing of demands in the network;
- minimise the objective function, which is given as a sum of costs of all actually installed links and nodes.

The discussed topological network design problem is generic. It may be interpreted as a task of optimising the topology of a backbone network. Such a design task is a difficult combinatorial optimisation problem and is known to be NP-hard. As the use of exact methods is, in this case, limited to very simple network examples, the heuristic approach has to be considered. Moreover, due to the local optima problem, only more sophisticated methods can prove to be effective.

The paper provides a mathematical formulation of the considered problem and proposes a set of adequate heuristic algorithms. The presented approaches encompass original ideas as well as select methods described in literature and their enhancements.

The proposed algorithms can deal with problems where link cost functions are either linear functions of link capacity with a fixed cost (e.g. link cost may depend on the geographical distance between nodes and a fixed link installation fee) or concave functions of link capacity (reflecting the economy of scale phenomenon common in telecommunication systems).

Numerical examples considered in this paper follow an assumption that the demands are directed and links are undirected (link capacity is taken as a sum of flows in both directions). However all of the algorithms could be easily adapted to solve problems with directed links if required. Similarly sets of transit nodes and access nodes are basically assumed to be disjoint but the algorithms allow for access nodes with transiting capability (mixed functionality).
The paper is organised as follows. Section 2 of the paper provides a mathematical formulation of the considered problem. Next, in Section 3, a range of specialised heuristic algorithms for topological network design is described in detail. Efficiency of each of the proposed methods is tested on a set of numerical examples – the network examples, obtained results and calculation times are presented and discussed in Section 4. Finally, the concluding remarks can be found in Section 5.

2. Problem formulation

Below, a general formulation of the transit node and link localisation problem is given. The link localisation problem, which is a special case of TNLLP, can be obtained by assuming a fixed transit node configuration and setting null transit node installation costs.
The TNLLP formulation presented below uses the link-path notation. The node-link notation is also available (c.f. [1]) and may be more suitable for MIP solvers.
As mentioned above, link cost can be given either by a linear or a concave function of link capacity. In order to obtain a MIP formulation, the $c_e y_e$ term should be used in (1) instead of $f_e(y_e)$, which allows for a nonlinear case.

TNLLP (link-path formulation)

indices

$d = 1, 2, \dots, D$	demands
$j = 1, 2, \dots, J_d$	paths for flows realising demand d
$e = 1, 2, \dots, E$	links
$v = 1, 2, \dots, V$	transit nodes

constants

h_d	volume of demand d
a_{edj}	1 if link e belongs to path j realising demand d, 0 otherwise
k_e	fixed link e installation cost
$f_e(y_e)$	variable cost of link e (a function of load y_e of link e)
b_{ev}	1 if link e is incident with transit node v, 0 otherwise
l_v	fixed transit node v installation cost
Y_e	upper bound of the capacity of link e (not active)
G_v	upper bound of the degree of transit node v (inactive)

variables

x_{dj}	flow realising demand d allocated to path j (non-negative continuous variable)
y_e	capacity of link e (non-negative continuous variable)
σ_e	1 if link e is provided, 0 otherwise (binary variable)
ε_v	1 if node v is provided, 0 otherwise (binary variable)

objective

$$\min \sum_e \left(f_e(y_e) + k_e \sigma_e \right) + \sum_v l_v \varepsilon_v \tag{1}$$

constraints

$$\sum_j x_{dj} = h_d \quad d = 1, 2, \dots, D, \tag{2}$$

$$\sum_d \sum_j a_{edj} x_{dj} = y_e \quad e = 1, 2, \dots, E, \tag{3}$$

$$y_e \le Y_e \sigma_e \quad e = 1, 2, \dots, E, \tag{4}$$

$$\sum_e b_{ev} \sigma_e \le G_v \varepsilon_v \quad v = 1, 2, \dots, V. \tag{5}$$

Constraints (2) guarantee realisation of demands imposed on the network. Constraints (3) and (4) ensure that zero capacity is assigned to all links that are not provided. Consistent link and node allocation is enforced by constraints 5). In the considered problems parameters Y_e and G_v are assumed to be inactive – they do not limit the capacity of edges and the degree of the nodes, respectively.
Localisation of access nodes is determined by the set of demands imposed on the network. Null installation fee is assumed for these nodes, since it does not have any influence on the produced solutions (other than shifting all of the results by a fixed value).

3. The methods

All of the considered heuristic methods and their modifications are presented below. A short description is provided in each case.

3.1. Flow shifting and rerouting

Most of the proposed flow shifting methods are new implementations of approaches already presented in literature (c.f. [1, 2]), in many places original modifications have been introduced. The flow shifting algorithms are rather simple and serve as basis for some of the more complex methods described in the following subsections.

3.1.1. Individual flow shifting

Individual flow shifting (IFS) was proposed in [1] as a simple heuristic for solving the LLP and TNLLP.
Firstly an initial allocation of demands is performed with a greedy type of algorithm. Subsequent demands are taken

in a random order and allocated to shortest paths. The paths are computed according to incremental cost of routing the demand in question on a given link (in particular if a link is empty its installation cost is added, otherwise only the variable cost is counted).
Secondly, in the main loop of IFS, subsequent demands are checked in a random order and reallocated if their rerouting results in a decrease of the total network cost.
The algorithm stops when the flow rerouting procedure can achieve no further network cost improvement.

3.1.2. Minoux greedy algorithm

The algorithm proposed by Minoux in [2] dealt with the LLP (ONDP). Here the Minoux greedy algorithm (MGA) is applied to the TNLLP.
Firstly initial allocation of demands is done, which may proceed as described for IFS.
Each iteration of the MGA main loop consists in assigning restoration costs δ_e to all of the links carrying some load. The δ_e value for a given link e is calculated as the cost of locally restoring the capacity of the link to an alternative shortest path. The link with the lowest negative restoration cost δ_e is switched off and its flows are redirected.
The algorithm stops when, in a given iteration, all of the links have a positive restoration cost δ_e, meaning that no further improvement of the total network cost can be achieved.

3.1.3. Accelerated Minoux greedy algorithm

The accelerated Minoux greedy algorithm (aMGA) [2] is similar to MGA, however here changes of restoration costs δ_e for subsequent iterations are assumed to be monotonic – hence it is no longer necessary to recalculate all of the δ_e in each iteration. Although the above assumption may not be true for all types of link cost functions, aMGA can be used in all cases and usually provides the same results as MGA while requiring fewer relocation cost computations (which, in turn, makes it significantly faster).

3.1.4. Bulk flow shifting

Initial allocation of demands is performed as described for IFS.
In the main bulk flow shifting (BFS) loop, the algorithm looks through all of the links in a random order. The currently analysed link is switched off and demands crossing it are disconnected. Subsequently an attempt is made to reallocate these demands to new paths. If the reallocation is not feasible or causes an increase of network cost then rollback to the previous configuration is performed.
The algorithm stops when no further improvements are achievable.
In the case of MGA flows from the removed link were rerouted locally and jointly. In case of BFS they are rerouted globally and individually which usually produces a better result. This algorithm differs from the bulk flow shifting approach presented in [1] where the deallocation decision was based on δ_e computations as described for MGA.
Several variants of the algorithm have been analysed. The variant described above is marked as BFS/1. BFS/n stands for a similar method where flows are deallocated on per node basis rather than per link basis. In BFS/b all of the nodes and links are checked in each iteration and the best (not the first better) option is chosen for switch off.
Another option to consider is whether to switch appropriate links/nodes permanently off or rather allow for flow allocation on these links/nodes in the following iterations. These options are indicated as BFS.off and BFS.on, respectively.

3.2. Yaged algorithm

The Yaged algorithm (YAG) proposed in [3] has been used to find locally optimal solutions of network design problems characterised by concave link cost functions over a set of linear flow constraints. The method is known to converge, in a finite number of steps, to local optimum points compliant with the Kühn-Tucker conditions.
Extensions of the algorithm have been presented in [1]. Here the method is applied to the TNLLP.
First an initial allocation of demands is done.
Subsequent iterations of the main YAG loop consist in recalculation of shortest paths for all of the demands with marginal link costs ($\mathrm{d}f_e(y_e)/\mathrm{d}y_e$) taken as link weights for the Dijkstra algorithm.
The algorithm stops when a fixed point is reached, i.e. two subsequent iterations yield exactly the same flow pattern.

3.3. The notion of adaptive function loop

The adaptive function loop (AFL) approach has been used in [1], combined with various versions of the Yaged algorithm, to deal with design problems characterised by concave link cost functions (requiring global optimisation). Similar cost smoothing techniques were mentioned in [1].
The idea consists in placing any of the simple methods (SM) such as IFS, MGA, aMGA, BFS or YAG in an outer loop called the adaptive function loop. The AFL modifies the link cost function for consecutive SM runs. The idea is to partially linearise the link cost function according to the following formula:

$$C_e(y_e) = \begin{cases} f_e(y_e) & \text{for } y_e \geq y^{th} \\ \beta_e^{th} \cdot y_e & \text{for } y_e < y^{th} \end{cases}$$

y_e – load of link e
$f_e(y_e)$ – original cost function of link e
y^{th} – threshold link load for linearisation
$\beta_e^{th} = f_e(y^{th})/y^{th}$

In subsequent AFL iterations y^{th} value is gradually decreased, hence the first iteration is performed with a linear link cost function (high y^{th} value) and the final one with the original link cost function ($y^{th} = 0$). Solution obtained by SM in each AFL step serves as starting point for the next iteration (Fig. 1).

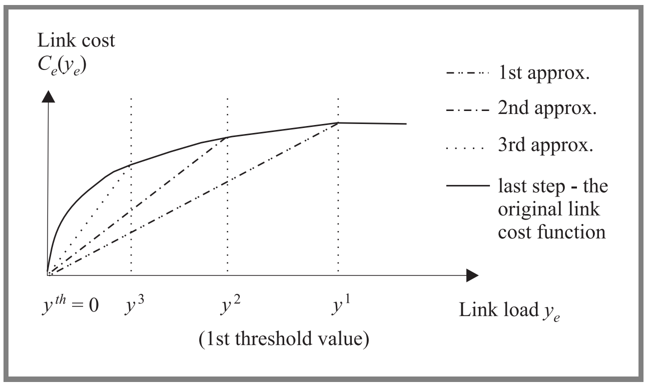

Fig. 1. Cost function linearisation concept in AFL.

The AFL approach has two advantages. Due to the linearisation, the fixed cost is partially included in computations of relocation gain (this tends to give better results). Moreover, by changing the initial y^{th} value and/or the number of adaptive steps n^{adapt}, the algorithm may be directed to various areas of the solution space, thus dealing with the local minimum problem.

Since it is not possible to determine y^{th} and n^{adapt} values that would work equally well for all kinds of problems, performing multiple AFL loop runs for various combinations of these parameters seems to be a good idea. Of course such an approach results in a proportional increase of computation time.

As mentioned above the AFL approach can be combined with various simple methods. The following options may be formed:

Individual flow shifting with adaptive function loop (**IFS-AFL**)
Minoux greedy algorithm with adaptive function loop (**MGA-AFL**)
Accelerated Minoux greedy algorithm with adaptive function loop (**aMGA-AFL**)
Bulk flow shifting with adaptive function loop (**BFS-AFL**)
Yaged algorithm with adaptive function loop (**YAG-AFL**)

3.4. Simulated allocation

The simulated allocation (SAL) algorithm (c.f. [5]) has been already applied to the TNLLP in [1]. Here a simple version of the method along with some enhancements has been implemented for comparison with other methods.

The simulated allocation works with partial allocation states. In each step it decides either to allocate or deallocate a randomly chosen demand (allocation is chosen with a higher probability in order to proceed towards full allocation states). Current value of α factor (defined as fraction of allocated flows) regulates the probability of allocation P_a and deallocation P_d. For example, for $\alpha < 0.8$ use $P_a = 1$ and for $\alpha \geq 0.8$ use $P_a = 0.7$.

Every now and then, when a complete allocation state is reached, a bulk deallocation of demands is performed (e.g. half of all the demands are deallocated). Bulk deallocation enables the algorithm to explore the solution space. Best complete allocation state that is reached is this manner is stored as the final solution.

The algorithm stops if the result is not improved for a predefined number of consecutive full allocation states.

One of the introduced enhancements consists in the way the bulk deallocation is performed. Instead of operation on per flow basis, the procedure deallocates all flows from randomly chosen links or nodes until the requested value of $\alpha = 0.5$ is reached. An on/off approach, similar to the one introduced in BFS.on and BFS.off, was also attempted but did not prove to be effective.

3.5. Genetic algorithm

The proposed genetic algorithm (GA) is based on the $(\mu + \lambda)$ evolution strategy [5]. It involves the use of problem-specific encoding and genetic operators.

The population consists of chromosomes that represent various structures of the network. Each chromosome is built of E binary genes corresponding to each link e of the network. When a link is allocated in a given network structure, the corresponding gene value of the considered chromosome is set to 1. Genes set to 0 indicate not allocated links.

The initialisation procedure builds the initial parent population P_0 by creating μ chromosomes. Half of the P_0 is generated by the greedy initialisation procedure already described for IFS, where shortest paths for the demands are computed in a random order, thus providing diverse solutions. The other half is derived by routing the demands on shortest paths obtained for randomly generated link metrics. In this manner diversity is introduced into P_0.

Inside the main program loop an offspring population O_n consisting of λ chromosomes is formed by copying randomly chosen chromosomes from parent population P_n and mutating them. Mutation consists in introduction of gene variations corresponding to changes in the structure of the network. The following mutation operators may be applied to randomly chosen links and/or nodes: node switch on/off, single link switch on/off and multiple links switch on/off. Each of these actions is performed with a defined probability. After mutation, O_n is evaluated (the demands are routed in the derived network structure and the total network cost is calculated for each chromosome) and the algorithm proceeds to the selection procedure. Here μ best-fitted (i.e. characterised by a minimal network cost) chromosomes are chosen and they become the parent population P_{n+1} for iteration $n+1$.

The loop is exited and the program ends when one of the following two conditions occurs: the algorithm is unable to improve the current best solution for a specified number of

Table 1
Parameters of the tested networks

Network	Number of transit nodes	Number of access nodes	Number of links	Number of demands	min h_d	max h_d	$\sum_d h_d$
N7	7	5	90	42	240	1 920	34 320
N14	14	11	418	182	120	7 560	172 320
N28	28	15	1 050	756	120	30 240	892 492
N9	9	19	132	171	30	150	14 220

generations (so called patience) or it exceeds the maximal allowable number of generations.
The GA version described above is rather simple and there is a lot of room for further enhancements. Implementation of crossover operators could be considered. Explorative properties of the algorithm should be analysed versus its ability to find locally optimal solutions – in this way values of the μ and λ parameters as well as mutation probability could be tuned.

4. Numerical results

Effectiveness of all of the presented methods has been tested on a range of numerical examples of diverse complexity. The algorithms have been implemented in ANSI C and executed on a PC equipped with Athlon 900 MHz processor and 256 MB of RAM.

4.1. The example networks

Three example network structures characterised by different number of nodes have been analysed: N7, N14 and N28 (available from [7]). Link costs were assumed to be linear with a fixed installation cost. Another example network N9 was used with concave link cost functions. The basic parameters of the networks are given in Table 1. All links are potentially available.
Additionally several variants of N7, N14 and N28 were introduced in order to analyse a range of fixed and variable link cost relations (for LLP and TNLLP) as well as the influence of various node installation costs (for TNLLP).
The unit cost c_e of link e is proportional to its geographical length. The fixed installation cost is given by $k_e = c_e \cdot 10^n$. For LLP analysis $n \in \{0,2,3,6\}$, whereas for TNLLP analysis $n \in \{4,5\}$ (and fixed costs of links are additionally multiplied by 3 for access links and by 2 for transit links). For TNLLP, the fixed installation cost of transit nodes is given by $l_v = 10^k$, where $k \in \{4,5,6\}$. There is no installation fee for access nodes.
Results obtained for all of the algorithms, along with the computation times, are summarised in the tables presented in subsections 4.2 and 4.3. In order to draw conclusions on algorithm average performance 10 runs of each of the algorithms have been conducted – each with a different seed value for the randomiser.
Bold font indicates the cases when the optimal solution or the best suboptimal solution has been achieved. Due to the problem complexity, an exact solution (taken from [1]) is available only for the smallest networks and is given in the EX row. It is worth noting that for all of the considered network examples the BFS-AFL algorithm has managed to provide either the optimal solution or the best suboptimal solution.

4.2. TNLLP results

Tables 2–5 refer to the TNLLP problem where link cost function are linear with a fixed installation cost. Table 2 contains the average solutions from 10 runs. The best solutions obtained with each method are noted in Table 3. Table 4 shows the relation between the average solution and the best one known (given by (*average – best_known)/best_known*). Average calculation times are gathered in Table 5.
BFS-AFL algorithms provided the best average results. BFS/l.on variant proved to be especially effective. SAL.on also performed well as far as the average results are concerned – especially for the simpler networks.
Optimal results or best suboptimal results were also achieved by simple algorithms from the BFS group, however their average results were somewhat worse. Taking into consideration short calculation times for algorithms of this kind, a random approach consisting in performing a number of runs and choosing the best solution could be applied in this case.
Effectiveness of the considered methods is well depicted by the "relative distance from the best" presented in Table 4 and the computation times from Table 5.
The simple algorithms (IFS, MGA, aMGA, BFS and YAG) are characterised by very short computation times, however they produce solutions way below expectations. The only significant exception here is the BFS algorithm group, which requires only slightly more time to provide already acceptable solutions. Especially, the BFS/b.on variant can be considered as a good alternative to more complex and time-consuming algorithms such as SAL and GA.
Introduction of the AFL loop has significantly improved the results provided by most of the simple methods. The AFL loop worked very well with IFS, BFS and YAG. The results for MGA, with or without AFL, are disappointing for TNLLP. For IFS and YAG a huge improvement, as

Table 2
TNLLP: average results (10 runs average) [cost unit 10^6]

Network	N14						N28					
n	4			5			4			5		
k	4	5	6	4	5	6	4	5	6	4	5	6
IFS	80.16	81.15	91.05	342.10	343.09	352.99	298.21	299.56	313.06	904.80	906.15	919.65
MGA	78.39	79.38	89.28	309.94	310.93	320.83	306.36	307.71	321.21	861.10	862.45	875.95
aMGA	78.41	79.40	89.30	311.52	312.51	322.41	306.36	307.71	321.21	861.90	863.25	876.75
BFS/l.off	58.96.	59.84	69.36	271.03	271.98	281.43	270.79	272.07	287.04	742.13	743.46	756.69
BFS/l.on	59.35	60.34	70.24	266.16	267.15	277.05	263.29	264.64	278.14	742.74	744.09	757.59
BFS/n.on	59.07	60.06	69.96	267.56	268.55	278.45	263.02	264.37	277.87	748.91	750.26	763.76
BFS/b.on	58.49	59.48	69.38	265.49	266.48	276.38	258.54	259.89	273.39	735.17	736.52	750.02
YAG	80.16	81.15	91.05	342.10	343.09	352.99	298.21	299.56	313.06	904.80	906.15	919.65
IFS-AFL	61.94	63.10	76.67	334.77	335.76	349.52	254.25	256.63	274.03	774.74	782.42	786.01
MGA-AFL	78.58	79.57	89.47	306.21	307.20	317.10	302.58	303.93	317.43	877.32	878.67	892.17
aMGA-AFL	78.58	79.57	89.47	306.21	307.20	317.10	302.58	303.93	317.43	877.32	878.67	892.17
BFS/l.off-AFL	57.86	58.82	**67.14**	265.81	266.80	276.52	258.16	260.31	273.93	730.33	731.77	745.87
BFS/l.on-AFL	**57.61**	**58.60**	**67.14**	**265.32**	**266.31**	**276.21**	**250.78**	**252.24**	**265.96**	**725.13**	**726.48**	**741.12**
BFS/n.on-AFL	**57.61**	**58.60**	67.60	**265.32**	**266.31**	**276.21**	256.13	257.28	271.99	727.67	728.62	742.70
YAG-AFL	63.28	64.27	74.17	338.63	339.62	349.52	254.92	256.27	269.77	782.58	783.93	797.43
SAL.on	**57.61**	**58.60**	68.50	**265.32**	**266.31**	**276.21**	262.38	263.73	277.23	738.74	740.09	753.59
SAL.off	65.23	65.87	72.06	306.00	306.79	314.67	287.64	288.72	298.37	845.65	846.76	857.70
GA	57.61	58.60	67.67	265.59	266.58	276.78	261.57	263.73	276.04	746.60	747.86	760.04
EX	**57.61**	**58.60**	**67.14**									

Table 3
TNLLP: best results (10 runs data) [cost unit 10^6]

Network	N14						N28					
n	4			5			4			5		
k	4	5	6	4	5	6	4	5	6	4	5	6
IFS	70.29	71.28	81.18	295.74	296.73	306.63	272.39	273.74	287.24	819.12	820.47	833.97
MGA	68.12	69.11	79.01	287.33	288.32	298.22	283.71	285.06	298.56	780.98	782.33	795.83
aMGA	68.12	69.11	79.01	288.30	289.29	299.19	283.71	285.06	298.56	780.98	782.33	795.83
BFS/l.off	**57.61**	**58.60**	**67.14**	**265.32**	**266.31**	**276.21**	261.60	262.86	275.46	733.00	734.26	746.86
BFS/l.on	**57.61**	**58.60**	68.50	**265.32**	**266.31**	**276.21**	255.81	257.16	270.66	729.96	731.31	744.81
BFS/n.on	**57.61**	**58.60**	68.50	**265.32**	**266.31**	**276.21**	259.79	261.14	274.64	733.14	734.49	747.99
BFS/b.on	**57.61**	**58.60**	68.50	**265.32**	**266.31**	**276.21**	254.79	256.14	269.64	726.48	727.83	741.33
YAG	70.29	71.28	81.18	295.74	296.73	306.63	272.39	273.74	287.24	819.12	820.47	833.97
IFS-AFL	59.85	60.02	71.21	304.49	305.48	315.38	252.31	255.60	270.99	747.33	748.68	756.98
MGA-AFL	68.12	69.11	79.01	287.33	288.32	298.22	283.71	285.06	298.56	779.85	781.20	794.70
aMGA-AFL	68.12	69.11	79.01	287.33	288.32	298.22	283.71	285.06	298.56	779.85	781.20	794.70
BFS/l.off-AFL	**57.61**	**58.60**	**67.14**	**265.32**	**266.31**	**276.21**	255.76	258.60	272.98	726.31	727.66	741.16
BFS/l.on-AFL	**57.61**	**58.60**	**67.14**	**265.32**	**266.31**	**276.21**	**250.11**	**251.74**	**265.02**	**723.54**	**724.89**	**738.23**
BFS/n.on-AFL	**57.61**	**58.60**	**67.14**	**265.32**	**266.31**	**276.21**	254.01	255.79	271.13	726.31	727.66	741.16
YAG-AFL	58.93	59.92	69.82	304.49	305.48	315.38	253.87	255.22	268.72	761.03	762.38	775.88
SAL.on	**57.61**	**58.60**	68.50	**265.32**	**266.31**	**276.21**	257.93	259.28	272.78	731.47	732.82	746.32
SAL.off	63.21	63.93	70.69	285.20	286.10	295.10	275.61	276.69	287.49	801.09	802.26	813.96
GA	**57.61**	**58.60**	**67.14**	**265.32**	**266.31**	**276.21**	257.14	261.52	271.70	731.62	732.97	750.79
EX	**57.61**	**58.60**	**67.14**									

Table 4
TNLLP: average – best_known/best_known [%]

Network	N14						N28					
n	4			5			4			5		
k	4	5	6	4	5	6	4	5	6	4	5	6
IFS	39	38	36	29	29	28	19	19	18	25	25	25
MGA	36	35	33	17	17	16	22	22	21	19	19	19
aMGA	36	35	33	17	17	17	22	22	21	19	19	19
BFS/l.off	2	2	3	2	2	2	8	8	8	3	3	3
BFS/l.on	3	3	5	0	0	0	5	5	5	3	3	3
BFS/n.on	3	2	4	1	1	1	5	5	5	4	3	3
BFS/b.on	2	1	3	0	0	0	3	3	3	2	2	2
YAG	39	38	36	29	29	28	19	19	18	25	25	25
IFS-AFL	8	8	14	26	26	27	2	2	3	7	8	6
MGA-AFL	36	36	33	15	15	15	21	21	20	21	21	21
aMGA-AFL	36	36	33	15	15	15	21	21	20	21	21	21
BFS/l.off-AFL	0	0	0	0	0	0	3	3	3	1	1	1
BFS/l.on-AFL	0	0	0	0	0	0	0	0	0	0	0	0
BFS/n.on-AFL	0	0	1	0	0	0	2	2	3	1	1	1
YAG-AFL	10	10	10	28	28	27	2	2	2	8	8	8
SAL.on	0	0	2	0	0	0	5	5	5	2	2	2
SAL.off	13	12	7	15	15	14	15	15	13	17	17	16
GA	0	0	1	0	0	0	5	5	4	3	3	3

Table 5
TNLLP: calculation times (10 runs average) [s]

Network	N14						N28					
n	4			5			4			5		
k	4	5	6	4	5	6	4	5	6	4	5	6
IFS	0	0	0	0	0	0	0	1	1	0	0	1
MGA	0	0	0	0	0	0	0	0	0	1	0	0
aMGA	0	0	0	0	0	0	0	0	1	0	0	0
BFS/l.off	0	1	0	0	0	0	5	5	5	5	5	5
BFS/l.on	0	0	0	1	0	0	5	5	5	4	4	4
BFS/n.on	0	0	0	0	0	0	5	5	5	4	4	4
BFS/b.on	1	1	1	1	1	1	43	43	43	23	22	23
YAG	0	0	0	0	0	0	1	1	1	0	1	1
IFS-AFL	9	9	8	6	6	6	131	134	122	104	104	105
MGA-AFL	1	1	1	1	1	1	4	4	4	5	5	5
aMGA-AFL	1	1	1	1	1	1	4	4	3	4	4	4
BFS/l.off-AFL	41	40	35	41	42	41	708	708	725	736	732	737
BFS/l.on-AFL	45	45	39	42	42	42	762	785	915	760	762	767
BFS/n.on-AFL	48	49	36	37	37	37	956	934	751	952	968	946
YAG-AFL	6	6	6	4	4	4	87	87	87	71	71	71
SAL.on	9	9	9	9	9	9	94	94	94	117	118	117
SAL.off	9	9	9	8	8	8	84	84	92	88	88	88
GA	44	48	80	48	48	48	872	828	951	855	840	831

Table 6
LLP: results (single run) [cost unit 10^6]

Network	N7				N14				N28			
n	0	2	3	6	0	2	3	6	0	2	3	6
IFS	**3.76**	3.95	5.22	1077.71	**26.34**	27.32	33.16	2286.59	151.24	154.81	175.40	5433.15
MGA	**3.76**	**3.94**	5.25	488.57	**26.34**	27.26	32.74	1294.16	151.24	154.74	176.48	2623.15
aMGA	**3.76**	3.95	5.25	488.57	**26.34**	27.33	32.96	1382.16	151.24	154.84	177.31	3281.58
BFS	**3.76**	**3.94**	5.12	503.71	**26.34**	27.12	31.44	1119.19	151.23	154.17	168.58	3035.84
YAG	**3.76**	3.95	5.22	1077.71	**26.34**	27.36	33.37	2358.59	151.24	154.87	176.04	5433.15
IFS-AFL	**3.76**	**3.94**	5.10	**460.69**	**26.34**	27.13	30.58	1261.56	151.24	154.48	167.97	3018.90
MGA-AFL	**3.76**	**3.94**	5.25	488.57	**26.34**	27.26	32.74	1210.20	151.24	154.71	175.88	2458.68
aMGA-AFL	**3.76**	**3.94**	5.25	488.57	**26.34**	27.26	32.74	1138.20	151.24	154.70	176.02	2491.64
BFS-AFL	**3.76**	**3.94**	5.03	**460.69**	**26.34**	27.05	**30.08**	**1062.01**	**151.23**	**153.91**	**165.09**	**2371.12**
YAG-AFL	**3.76**	**3.94**	5.10	548.44	**26.34**	27.36	31.41	1435.27	151.24	154.87	175.57	3114.65
SAL	**3.76**	**3.94**	5.16	**460.69**	**26.34**	27.14	31.47	1172.97	151.23	154.26	169.43	2504.49
EX	**3.76**	**3.94**	**5.00**	**460.69**	**26.34**	**27.02**	**30.08**					

Table 7
LLP: calculation times (single run) [s]

Network	N7				N14				N28			
n	0	2	3	6	0	2	3	6	0	2	3	6
IFS	0	0	0	0	0	0	0	0	1	1	1	1
MGA	0	0	0	0	0	0	0	0	1	1	1	1
aMGA	0	0	0	0	0	0	0	0	1	0	1	0
BFS	0	1	0	0	0	0	0	0	3	4	7	3
YAG	0	0	0	0	0	0	0	0	1	1	0	0
IFS-AFL	1	1	1	1	25	27	27	23	370	383	378	340
MGA-AFL	0	0	0	0	8	10	6	4	129	157	69	51
aMGA-AFL	1	0	1	0	6	4	2	2	39	23	14	11
BFS-AFL	2	2	3	2	52	71	71	71	966	1033	1051	1203
YAG-AFL	0	1	0	0	4	4	5	3	54	55	57	54
SAL	1	0	0	1	4	3	4	6	43	69	55	71

compared to the simple methods, has been achieved while keeping the calculation times pretty low. BFS-AFL approach provided clearly the best results of all the algorithms, being at the same time the most time consuming. Executing the BFS-AFL with fewer AFL iterations, resulted in proportionally reduced calculation times and slightly worse solutions (comparable with methods such as SAL).

As already mentioned in Section 3.3, the algorithms perform multiple AFL runs for different combinations of initial threshold parameter and number of steps parameter. This approach allows for better exploration of the solution space, but results in proportionally longer computations. In this numerical study each of the algorithms performed 40 AFL iterations for TNLLP and 120 for LLP.

The simple implementation of the SAL algorithm also proved to be a very good performer. It provided results at least as good as those cited in [1]. However attempts to further improve the solutions produced by SAL by tuning of the algorithms parameters, even at the cost of significant extensions of the computation time, did not produce any substantial gain.

The performance of GA should be regarded as promising, given the fact that the implemented version of the algorithm requires further tuning and study. However, it should be kept in mind that genetic algorithms are usually quite slow.

4.3. LLP results

The discussed algorithms were tested also on LLP network examples. A single run of all of the methods was performed. The results and computation times are presented in Tables 6 and 7, respectively.

The relative performance of the algorithms as well as their basic characteristics are comparable to those observed for TNLLP and commented in the previous subsection. Also in this case, the BFS-AFL provided the best results. MGA and especially aMGA worked much better with LLP than with TNLLP.

4.4. Concave link cost functions

Network example N9 has been used in order to present the capability of the studied algorithms to deal with problems characterised by concave functions. Link cost function of the following form was assumed (with $A = 0.05$, $B = 300$ and $C = 0.003$):

$$f(y_e) = (-\exp(-Cy_e) + 1) \cdot (Ay_e + B) . \qquad (6)$$

The results are summarised in Table 8. The average values are derived from ten runs of each of the algorithms.

Table 8
TNLLP: N9 – concave function example (10 runs data)

Algorithm	Average cost	Best cost	Avg.-best /best [%]	Time [s]
IFS	14733.97	14313.07	5.2	0
MGA	15571.60	14507.29	11.1	0
AMGA	14398.66	14114.22	2.8	0
BFS/l.off	14258.21	14060.45	1.8	0
BFS/l.on	14180.96	14065.07	1.2	1
BFS/n.on	14619.52	**14010.17**	4.3	0
BFS/b.on	14031.60	**14010.17**	0.2	1
YAG	14733.97	14313.07	5.2	0
IFS-AFL	14221.04	14037.11	1.5	5
MGA-AFL	15553.69	14507.29	11.0	1
aMGA-AFL	14368.22	14114.22	2.6	0
BFS/l.off-AFL	14021.05	**14010.17**	0.1	13
BFS/l.on-AFL	14104.12	**14010.17**	0.7	28
BFS/n.on-AFL	**14010.17**	**14010.17**	0.0	26
YAG-AFL	14270.68	14037.11	1.9	2
SAL.on	14046.07	**14010.17**	0.3	8
SAL.off	14082.38	**14010.17**	0.5	8
GA	14055.06	**14010.17**	0.3	33

As expected, all of the algorithms are able to deal with the concave link cost problem. Again the BFS-AFL, SAL and GA are the most effective methods. This time it is the BFS/n.on-AFL variant that provided the best result in all of the iterations. Pretty good performance of IFS-AFL and YAG-AFL can be observed, especially as compared to the simple versions of these methods.
Longest calculation times were experienced for BFS-AFL algorithms and for GA.

5. Conclusions

The generic topological network design problem has been discussed. The link-path formulation of the design task has been presented in order to express the considered problem in a formal manner. Two subproblems, namely the LLP and the TNLLP, have been distinguished. Since exact solution methods are applicable only to trivial networks, adequate heuristic methods capable of providing good suboptimal solutions in reasonable time are called for.
A wide range of specialised heuristic algorithms has been presented. The considered solution methods encompass algorithms known from literature, their modifications and enhancements as well as original ideas. Efficiency of the proposed methods has been demonstrated on a set of numerical examples of diverse complexity. Most of the implemented algorithms proved to be suitable for solution of the considered network examples.
The numerical study enabled a comparison of the methods and selection of the most effective algorithms. The usual time versus quality trade-off has been observed. Simple methods based on the BFS principle were able to provide decent solutions in very short time. Slightly more time consuming versions of BFS provided results comparable to SAL and GA. SAL delivered quite good solutions in a reasonable time. BFS-AFL outperformed all of the other algorithms in terms on solution quality, however required significantly longer computation times. GA implementation was also very time consuming. Generally speaking the AFL notion proved very successful. Depending on the imposed time and result quality requirements, an appropriate algorithm from the discussed range can be selected.
Analysis of relative efficiency or the presented methods gives valuable insight into advantages and drawbacks of various heuristic approaches to solution of the topological design task. Of course the proposed algorithms are not perfect and there is a lot of room for improvements and tuning. Derivation of a good lower bound of the optimal solution would be helpful for algorithm evaluation purposes and/or as a stop criterion. Implementation of more effective procedures, e.g. for shortest paths computation, should be also considered.

References

[1] M. Pióro, A. Jüttner, J. Harmatos, Á. Szentesi, P. Gajowniczek, and A. Mysłek, "Topological design of telecommunication networks. Nodes and links localization under demand constraints" in *Proc. 17th Int. Teletraf. Congr.*, 2001.

[2] M. Minoux, "Network synthesis and optimum network design problems: models, solution methods and application", *Networks*, vol. 19, pp. 313–360, 1989.

[3] B. Yaged Jr., "Minimum cost routing for static network models", *Networks*, vol. 1, pp. 139–172, 1972.

[4] P. Karaś, S. Kozdrowski, and M. Pióro, "Doubly iterative algorithm for multi-layer network design" in *6th PSRT*, Oficyna Wydawnicza Politechniki Wrocławskiej, 1999.

[5] M. Pióro and P. Gajowniczek, "Solving multicommodity integral flow problems by simulated allocation", *Telecommun. Syst.*, vol. 7, no. 1–3, 1997.

[6] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. 3rd ed., Springer, 1996.

[7] Network examples can be downloaded from, http://www.tele.pw.edu.pl/networks/TNLLP/

Piotr Karaś received his M.Sc. degree in telecommunication in the Institute of Telecommunications at Warsaw University of Technology in 1999. His M.Sc. thesis concerned concave design problems related to multi-layer telecommunication networks. Currently, he is following Ph.D. studies at Warsaw University of Technology under the supervision of Prof. Michał Pióro. His Ph.D. thesis concerns use of heuristic algorithms in difficult network design tasks. Since 1999 he is working with PTC mobile operator as a specialist in network dimensioning. His research interests include various aspects of network design, core network architecture, traffic analysis and prediction.
pkaras@tele.pw.edu.pl
Institute of Telecommunications
Warsaw University of Technology
Nowowiejska st 15/19
00-665 Warsaw, Poland