Int. J. Appl. Math. Comput. Sci., 2006, Vol. 16, No. 4, 487–502

QUASI–HIERARCHICAL EVOLUTION ALGORITHM FOR FLOW ASSIGNMENT IN SURVIVABLE CONNECTION–ORIENTED NETWORKS

MICHAŁ PRZEWOŹNICZEK*, KRZYSZTOF WALKOWIAK**

* Faculty of Computer Science and Management, Wrocław University of Technology
ul. Wybrzeże Wyspiańskiego 27, 50–370 Wrocław, Poland
e-mail: Michal.Przewozniczek@pwr.wroc.pl

** Chair of Systems and Computer Networks, Faculty of Electronics, Wrocław University of Technology
ul. Wybrzeże Wyspiańskiego 27, 50–370 Wrocław, Poland
e-mail: Krzysztof.Walkowiak@pwr.wroc.pl

The main objective of this paper is to develop an effective evolutionary algorithm (EA) for the path-assignment problem in survivable connection-oriented networks. We assume a single-link failure scenario, which is the most common and frequently reported failure event. Since the network flow is modeled as a non-bifurcated multicommodity flow, the discussed optimization problem is NP-complete. Thus, we develop an effective heuristic algorithm based on an evolutionary algorithm. The main novelty of this work is that the proposed evolutionary algorithm consists of two levels. The "high" level applies typical EA operators. The "low" level is based on the idea of a hierarchical algorithm. However, the presented approach is not a classical hierarchical algorithm. Therefore, we call the algorithm quasi-hierarchical. We present its description and the results of simulation runs over various networks.

Keywords: evolution algorithm, optimization, connection-oriented networks

1. Introduction

In this paper we tackle the problem of path assignment in computer networks by an innovative evolutionary algorithm (EA). The optimization problem considered can be formulated as follows: Consider a network being a set of connections (commodities) defined by the triple: origin nodes, destination nodes and bandwidth requirements. The nonbifurcated flow assignment problem consists in routing the flow of commodities using only one route (path) for each commodity. Additionally, the capacity constraint must be satisfied, i.e., the flow of each arc (calculated as the sum of all commodities using that arc) cannot exceed the capacity of the arc. Therefore, we call a solution of the problem considered feasible if two conditions are satisfied. First, the whole flow for a given commodity must be routed (assigned) using only one path. Second, the total flow routed over a particular link must be within a given capacity limit. Examples of popular computer network techniques applying the connection-oriented (c-o) flow are the Asynchronous Transfer Mode (ATM) and MultiProtocol Label Switching (MPLS). Since the discussed problem is NP-complete (Karp, 1975; Pióro and Medhi, 2004), exact methods based on the branch-and-bound (BB) approach must be applied to find an optimal solution. According to (Pióro and Medhi, 2004), the application of BB methods for single path (non-bifurcated) flow allocation is limited to at most medium-size networks (of about 20 nodes) and, therefore, heuristic methods are of interest.

We focus on the survivability of c-o networks. The survivability of computer networks is very important since a loss of service means a loss of revenues, especially in high-speed fiber networks. A typical approach to provide survivability in c-o networks is as follows: Each circuit, i.e., a virtual path in the ATM or a label switched path in MPLS has a primary route and a backup route. The primary route is used for transmitting data in a normal, failure-free state of the network. When the primary route is broken, the failed circuit is switched to the backup route that uses a spare capacity. The process of switching is easy, i.e., the circuit's identifier numbers are changed in network nodes.

Most of the previous research on network survivability has concentrated on the Capacity and Flow Assignment (CFA) problem, in which link capacity and routing are optimized to provide full restoration after a failure (Grover, 2004; Pióro and Medhi, 2004). The CFA problem is applied in the network design phase or when the existing network is augmented. Typically, the objective of such problems is the combined cost of capacity and routing. In this work we consider an existing backbone

network. In many cases the network is in an operational phase and augmenting its resources (links, capacity) is not possible in a short time perspective. Consequently, the best method to provide survivability to an existing backbone network is the selection of primary paths that guarantees the allocation of flows enabling effective restoration of failed demands. A similar approach was applied in (Jæger and Tipper, 2003; Murakami and Kim, 1996; Nilsson *et al.*, 2003). The main problem is to find an objective function for primary route assignment that yields the preparation of the network to the rerouting process. Since, according to (Grover, 2004), the single-link failure is the most common and frequently reported failure event, we limit our deliberations to such a failure scenario. As the objective, we use the Lost Flow in Node (LFN) function. The LFN function can be applied for optimization of static and dynamic paths in MPLS and the ATM networks. The results presented in (Walkowiak, 2003; 2004a; 2004b; 2004c) show that applying various functions based on the LFN function can improve network survivability. For more information on network survivability, we refer the reader to (Grover, 2004; Pióro and Medhi, 2004) and the references given therein.

We propose an effective heuristic algorithm based on an evolutionary algorithm for the problem of primary route assignment in survivable connection-oriented networks with the LFN function as the objective. The main novelty of our approach is that the proposed algorithm consists of two levels. The "high" level applies typical EA operators. The "low" level idea is based on a hierarchical algorithm. Nevertheless, the presented approach is not a classical hierarchical algorithm. Thus, we call it quasi-hierarchical.

The paper is organized as follows: Section 2 surveys applications of genetic and evolutionary algorithms to various network problems. In Section 3, the optimization problem is formulated. Section 4 contains the description of the evolutionary algorithm. In Section 5, simulation results are presented and discussed. Finally, the last section concludes this work.

2. Related Work

Flow assignment and topological network design constitute topics of many publications. A significant number of them indicate possibilities of using algorithms based on breeding a population of solutions and affecting them by the well-known genetic operators. However, one can easily find publications containing a lot of propositions of GAs (genetic algorithms) or hybrid-GA algorithms trying to solve various flow assignment and network topological design problems that are much less complex than the one presented in this paper. First of all, most of them are typical GAs with bit-string notation, some are rather EAs (evolutionary algorithms) as their representations are integer-string-based, which entails tree (or any other graph) coding (Elbaum and Sidi, 1996; Riedl, 1998). A more advanced representation was also used in (Riedl, 2002) where gene values were associated with link weights in routing protocols, and in (White *et al.*, 1999), where a kind of hybrid representation was used (three different types of information were coded in one chromosome). In the above-mentioned publications, neither advanced GA construction, nor more specialized operators than crossing and mutation were used. We believe that by adding the presented "low" level of the algorithm, two main goals can be achieved, namely, higher search effectiveness and the possibility of using the algorithm for finding solutions in much larger search spaces than the ones examined in other publications. (For example, in (White *et al.*, 1999), only twenty-node networks were examined and the number of paths between node pairs was reduced to only eight shortest ones.)

A set of various evolutionary algorithms was presented in (Corne *et al.*, 2000). The objective was very close to the one considered here, i.e., network design. However, although a number of algorithms were presented and compared, and various operators were introduced (such as repair operators or advanced route crossing operators), algorithm construction is still a simple one and none of the algorithms was used for networks with more than 25 nodes (typically from 8 to 20).

3. Problem Formulation

We consider a computer network modeled as (G, c), where $G = (V, A)$ is a directed graph with n vertices representing routers or switches and m arcs representing links, and $c : A \rightarrow \mathbb{R}^+$ is a function that defines arc capacities. We denote by $o : A \rightarrow V$ and $d : A \rightarrow V$ functions defining respectively the origin and destination nodes of each arc. In our approach we assume that the bandwidth of various connections using the same link can be summed to check capacity constraints.

To mathematically represent the problem, we introduce the following notation: f_a is the flow of the arc a, c_a means the capacity of the arc a, $g_v^{\text{out}} = \sum_{i:o(i)=v} f_i$ is the aggregate flow in arcs leaving the node v, $e_v^{\text{out}} = \sum_{i:o(i)=v} c_i$ is the aggregate capacity of arcs leaving the node v.

Definition 1. The *global non-bifurcated m.c. flow*, denoted by $\underline{f} = [f_1, f_2, \ldots, f_m]$, is defined as a vector of flows in all arcs. We call a flow $\underline{f}$ *feasible* if the following inequality holds:

$$\forall a \in A : f_a \leq c_a. \tag{1}$$

We define the spare (residual) capacity of an arc as the difference between the capacity and the flow in that

arc. The spare capacity is used only for the purpose of rerouting failed connections.

For simplicity, we introduce the following function:

$$\varepsilon(x) = \begin{cases} 0 & \text{for} \quad x \leq 0, \\ x & \text{for} \quad x > 0. \end{cases} \tag{2}$$

To examine the main characteristics of local restoration, we consider an arc $a \in A$. We assume a failure of a. In local rerouting, the flow on the arc a must be rerouted by the source node of that arc. Therefore, the spare capacity of the outgoing arcs of $o(a)$ except k is a potential bottleneck of the restoration process. If the spare capacity of arcs leaving the node $o(a)$ is relatively small, a flow of the arc k could be lost. We define the function LA_a^{out} representing the flow loss in the node $o(a)$ in the following manner:

$$LA_a^{\text{out}}(\underline{f}) = \varepsilon\left(g_{o(a)}^{\text{out}} - (e_{o(a)}^{\text{out}} - c_a)\right). \tag{3}$$

Note that $LA_a^{\text{out}}(\underline{f})$ denotes a lost flow that cannot be restored using arcs leaving the node $o(k)$ due to limited spare capacities of these arcs. In other words, Eqn. (3) can be interpreted as a residual flow, if any, at the origin node of the arc a, available for rerouting connections using a after a failure of a. By analogy, we define the function LA_a^{in} of the lost flow that cannot be restored using arcs entering the node $d(a)$.

Notice that the value of the $LA_a^{\text{out}}(\underline{f})$ function depends on the flow $g_{o(a)}^{\text{out}}$ leaving the node $o(a)$, and it is not dependent directly on the flow in the arc a. Therefore, we formulate the function $LN_v^{\text{out}} : [0; e_v^{\text{out}}] \to [0; e_v^{\text{out}}]$ of the lost flow in the node v as the following sum over all arcs leaving that node:

$$\begin{aligned} LN_v^{\text{out}}(\underline{f}) &= \sum_{a:o(a)=v} \varepsilon\left(g_v^{\text{out}} - (e_v^{\text{out}} - c_a)\right) \\ &= \sum_{a:o(a)=v} LA_a^{\text{out}}(\underline{f}). \end{aligned} \tag{4}$$

Similarly, $LN_v^{\text{out}}(\underline{f})$ denotes the residual flow, if any, at the node v, available for rerouting connections using any arc leaving v.

Definition 2. We call an arc k *adjacent* to an arc a if $o(k) = o(a)$ or $d(k) = d(a)$. If k is not adjacent to the the arc a, we call it *remote* to a.

The functions $LA_a^{\text{out}}(\underline{f})$ and $LN_v^{\text{out}}(\underline{f})$ take account only of arcs adjacent to the failed arc a. Arcs remote to k, which can block some flow of the failed arc during the rerouting process, are not taken into account. Therefore, $LA_a^{\text{out}}(\underline{f})$ only estimates the flow of the arc k lost after local rerouting and, in much the same way, $LN_v^{\text{out}}(\underline{f})$ denotes the "potential" of the node v to make the rerouting of connections broken after a failure of any single arc leaving v.

Using $LA_a^{\text{out}}(\underline{f})$ and $LN_v^{\text{out}}(\underline{f})$, we can define the function $LFN(\underline{f})$ (lost-flow-in-node) that indicates the preparation of the whole network to local rerouting after a failure of any single arc:

$$LFN(\underline{f}) = \sum_{a \in A} LA_a^{\text{out}}(\underline{f}) = \sum_{v \in V} LN_v^{\text{out}}(\underline{f}). \tag{5}$$

We assume that the probabilities of arc failures are the same for all arcs. Therefore, no probabilities are included in this function.

Corollary 1. *$L_k^{\text{out}}(\underline{f})$, $LN_v^{\text{out}}(\underline{f})$ and $LFN(\underline{f})$ are continuous, nondecreasing, piecewise linear functions for any feasible flow $\underline{f}$. The function $LFN(\underline{f})$ is differentiable except for the points for which one of the following conditions holds*:

$$g_{o(a)}^{\text{out}} = (e_{o(a)}^{\text{out}} - c_a), \qquad \forall a \in A. \tag{6}$$

For more details and a comprehensive discussion of the function LFN, please refer to (Walkowiak 2004a; 2004b; 2004c).

We wish to underline that the concept of the LFN function was successfully applied to dynamic and static optimization of routes in MPLS and ATM networks (Walkowiak, 2004a; 2004b; 2004c).

Now we shall formulate the optimization problem of primary route assignment (PRA) with the objective function LFN given by (5). We assume that sets of candidate routes are given. To formulate the optimization problem, we use the following notation:

Sets:

- A set of m arcs (directed links),
- V set of n nodes (vertices),
- P set of connections (demands) in the network,
- Π_p index set of candidate routes for the connection p;

Indices:

- p connections (demands) in the network, used as subscript,
- k candidate routes, used as superscript,
- a arcs (directed links), used as subscript,
- v nodes, used as subscript;

Constants:

- δ_{pa}^k equal to 1 if the arc a belongs to the route k realizing the connection p or 0 otherwise,
- Q_p volume (estimated bandwidth requirement) of the connection p,
- c_a capacity of the arc a;

Variables:

x_p^k decision variable, which is 1 if the route $k \in \Pi_p$ is selected for the connection p and 0 otherwise,

f_a flow of the arc a.

The PRA optimization problem can be formulated as follows:

$$\min_{\underline{f}} LFN(\underline{f}), \tag{7}$$

subject to

$$\sum_{k \in \Pi_p} x_p^k = 1, \qquad \forall p \in P, \tag{8}$$

$$x_p^k \in \{0,1\}, \qquad \forall p \in P, \quad \forall k \in \Pi_p, \tag{9}$$

$$f_a = \sum_{p \in P} \sum_{k \in \Pi_p} \delta_{pa}^k x_p^k Q_p, \qquad \forall a \in A, \tag{10}$$

$$f_a \le c_a, \qquad \forall a \in A. \tag{11}$$

The objective function (7) is a flow lost in any node of the network due to a failure of any single link defined using the function given by (5). The function $LFN(\underline{f})$ is a sum over all links or all nodes in the network. Since we consider a single failure of any link and assume that the probability of such a failure is the same for all links, we do not introduce any probability into the function (5). Generally, the function $LFN(\underline{f})$ represents the preparation of the whole network to perform the restoration process using a local rerouting strategy for a failure of any single link. As has been mentioned above, in the local restoration the beginning node of the failed link is a bottleneck for the restoration. Therefore, it is very important to correctly design flows in links leaving that node. The condition (8) states that each connection can use only one primary route. The constraint (9) ensures that the decision variables are binary ones. Furthermore, (10) is the definition of a link flow. Finally, (11) denotes the capacity constraint. In the problem (7)–(11), we must find a route for each connection yielding a feasible non-bifurcated m.c. flow $\underline{f}$ that minimizes the value of the LFN function and satisfies the capacity constraint. It must be noted that the PRA problem is NP-complete (Karp, 1975; Pióro and Medhi, 2004). PRA is a 0–1 integer programming problem with linear constraints, which is generally considered very hard. The main reason for that is that the solution space that includes all possible paths is extremely large. The size of the problem is large even for relatively small networks.

A popular method to solve 0–1 problems is the branch-and-bound (B&B) approach. Such algorithms have been applied to many problems related to PRA (Kasprzak, 2003; Pióro and Medhi, 2004; Walkowiak 2004a). According to (Pióro and Medhi, 2004), application of BB methods to single path (non-bifurcated) flow allocation is limited to at most medium-size networks (of about 20 nodes), and therefore heuristic methods are of interest. The only way to solve the PRA problem by an exact algorithm that can produce the optimal solution is to reduce the size of the problem and consider only a part of all possible candidate routes. We call such a problem a *restricted* problem. This approach, called the path generation technique, was discussed in (Pióro and Medhi, 2004). However, in this work we wish to consider the PRA problem with the whole solution space, i.e., we do not introduce any constraints on candidate routes. We refer to such a problem as the *unrestricted* problem.

Several algorithms have been proposed to solve the PRA problem formulated above. In (Walkowiak, 2001; 2004a), a branch-and-bound algorithm and a genetic algorithm were presented for a restricted PRA problem. A heuristic algorithm FDP (Flow Deviation for Primary routes) based on the flow deviation (Fratta *et al.*, 1973) method was introduced in (Walkowiak, 2003; 2004b) for both restricted and unrestricted versions of the problem (7)–(11). Recently, one of the authors developed a heuristic called LRH (Lagrangean Relaxation Heuristic) that combines FDP and Lagrangian relaxation techniques (Walkowiak, 2006).

4. Algorithm Description

In this section we present the main characteristics and objectives of the evolutionary algorithm known as the HEFAN (Hierarchical Evolutionary algorithm for Flow Assigment in Non-bifurcated commodity flow), developed for the unrestricted PRA problem (7)–(11). All of the presented ideas are based on the approach set forth in the M.Sc. dissertation (Przewoźniczek, 2003). From a general point of view, the algorithm has only one objective: to find feasible and optimal (in terms of the fitness function) assignment of paths (routes) for all demands in an oriented network. A typical evolutionary system can easily meet this objective, but only if we provide a proper collection of routes for which the search will be done. Such a collection can be neither too poor, nor too rich. Indeed, a small collection will force us to search only within a small part of possible solution space. In turn, a large set of routes will make the problem intractable due to an excessive number of possibilities. These are the reasons why the presented system is conceptually divided into three abstract parts: a route proposal database, and "high" and "low" levels. The "high" level acts as a typical EA and the "low" one as an intelligent seeker for propositions of new routes, based on the EA idea using some of its features but actually not being such an algorithm. This part is organized as follows: first we describe the route database, and then the organization of the "high" and "low" levels. Finally, we shall present a general algorithm overview.

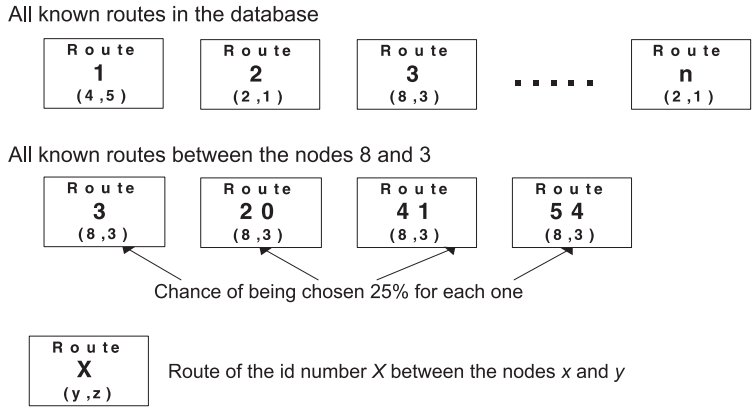

Fig. 1. Pick up mechanism for the database of routes.

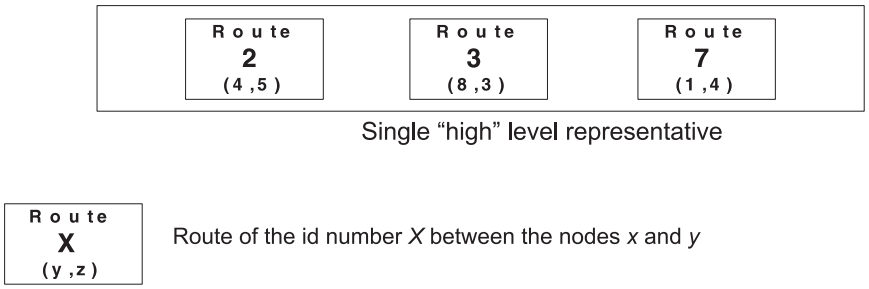

Fig. 2. High level individual representation.

It should be emphasized that the HEFAN was successfully applied to a network congestion problem in (Przewoźniczek and Walkowiak, 2005). However, the congestion problem differs strongly from the PRA problem addressed in this work. The objective function of PRA is separable with respect to arcs, i.e., LFN is a sum over all arcs of functions defined according to arc flows and capacities. The congestion problem considered in (Przewoźniczek and Walkowiak, 2005) consists in maximizing the minimal residual capacity of arcs. Therefore, it is a bottleneck problem, which is not separable with respect to arcs. Consequently, the evaluation of the HEFAN reported in (Przewoźniczek and Walkowiak, 2005) cannot be used to assess the performance of the HEFAN applied to the PRA problem.

4.1. Database of Routes. The database of known routes is one of the most important parts of the algorithm. It contains all known routes between the nodes existing in the network. The database may be requested to return a route proposal between a specified pair of nodes. The route is chosen randomly from the subset of routes having given origin and destination nodes. The chance for being chosen is equal for all routes (Fig. 1). It is possible that creating a nonequal chance-computing mechanism for route choosing is an option for a further improvement of the algorithm as some routes may fit better a specified set of routes than the others.

The start-up knowledge of the routes database is built from all possible routes whose lengths are smaller than l (a parameter set by the user). The database knowledge is also improved by route proposals from which an initial solution is built (if the user supported any). Also, if new route proposals have been created during an algorithm run, they can be added to the database and improve the knowledge already stored (this can also be seen in Fig. 10). There should be at least one route proposal between all possible node pairs in the database for proper work of the algorithm.

4.2. "High" Level of the Algorithm. The "high" level of the presented algorithm contains typical EA operators, which can be found in many publications and books on this topic (Davis, 1996; Kwaśnicka, 1998; Michalewicz, 1996). The representation of the problem considered is as follows: One chromosome is the set of route proposals for all commodities in the network. For instance, Chromosome 237 means that the first commodity uses Route 2, the second one uses Route 3, etc. An example of a high level individual is presented in Fig. 2.

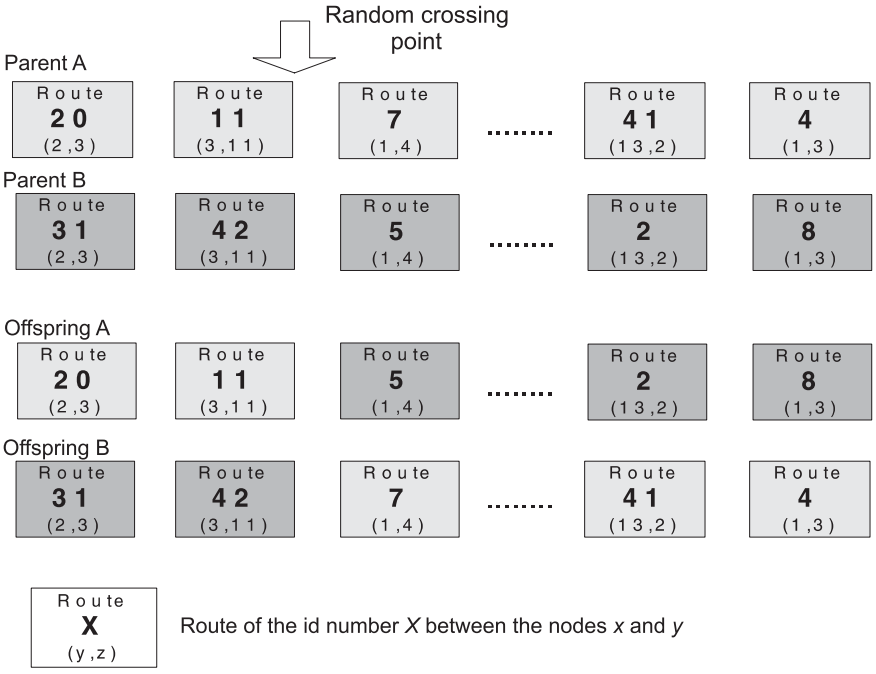

Fig. 3. High level one-point crossover operator.

The main operators of the "high" level are introduced in the following. The selection method is based on the roulette wheel technique. There are two crossover operators supported: the one-point crossover and the uniform crossover, selected randomly according to proportions set by the user. In the one-point crossover, we randomly pick up a crossing breakpoint and create offspring by copying the first x genes from the first parent and the rest from the second one. This operation is shown in Fig. 3.

The uniform crossover operator randomly copies genes from parents to offspring. If the first offspring in the n-th position has a gene from the first parent, then the second offspring will have in the same position a gene from the second parent, and vice versa. This operation is shown in Fig. 4.

The idea of using two different crossing operators is simple—the experiments were made for networks having from 10 to 36 nodes and demand matrices including demands for each source-destination node pair. These conditions give rather big (1260 for a 36-node network) numbers of genes for "high" level individuals (the number of paths that must be established in the network). In these terms, a typical order-dependent crossing operator might become ineffective as no inversion operator was supported. This situation forces adding either an inversion mechanism or a uniform crossover operator described above. The second possibility that was chosen in our investigations proved to be sufficient for good performance of the algorithm.

The mutation operator replaces a randomly picked gene (a path between the two nodes) by a different proposition of this gene (another route between the same nodes) from the database of routes.

The fitness function is based on the LFN function presented in (Walkowiak, 2003; 2004a). The only difference between LFN and the fitness used in the algorithm is that the algorithm's fitness function $F(i,p)$ for a single individual (chromosome) p in a generation i is defined as follows:

$$\mathrm{exc}(x) = \begin{cases} -x & \text{for} \quad x < 0, \\ 0 & \text{for} \quad x \geq 0, \end{cases}$$

$$PEN_v(\underline{f}) = \sum_{a:o(a)=v} \mathrm{exc}(LA_a^{\mathrm{out}}(\underline{f})),$$

$$LN'^{\mathrm{out}}_v(\underline{f}) = \begin{cases} LN_v^{\mathrm{out}}(\underline{f}) & \text{for} \quad PEN_v(\underline{f}) = 0, \\ (LN_v^{\mathrm{out}}(\underline{f}) + PEN_v(\underline{f}) * 10)^2 & \\ & \text{for} \quad PEN_v(\underline{f}) > 0, \end{cases}$$

$$LFN'(\underline{f}) = \sum_{v \in V} LN'^{\mathrm{out}}_v(\underline{f}),$$

$$F(i,p) = \frac{1}{LFN'(p)+1} + BN(i),$$

where $LFN'(p)$ is the value of the LFN function calculated for the route assignment given by the individual p. The difference between the LFN and LFN' functions is that the latter takes into account a penalty, if for a particular chromosome (route assignment), the capacity con-

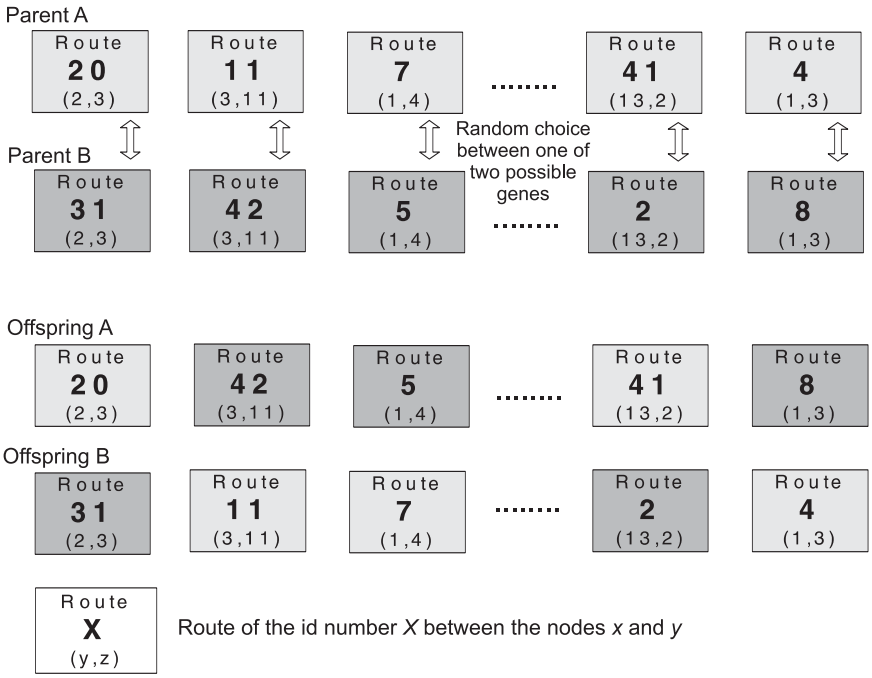

Fig. 4. High level uniform crossover operator.

straint (11) is violated (the flow through a link exceeds the capacity of that link). $BN(i)$ denotes the brainstorm modifier, which is discussed in what follows.

The *brainstorm* operator deals, for a period of time and in special way, with fitness function values for every individual and is a population state, rather than a typical operator. If the brainstorm is to work it must be first turned on under circumstances defined by the user. The operator is turned on if the average value of the fitness function for the last n generations is not greater than the same average value for the previous n generations increased by x%. Consequently, the brainstorm is turned on if

$$\frac{1}{n}\sum_{i=c-n}^{c} F_{\text{avg}}(i) \le \frac{1}{n}\sum_{i=c-2n}^{c-n} F_{\text{avg}}(i)\left(1+\frac{x}{100}\right),$$

where c is the number of the current generation and $F_{\text{avg}}(i)$ denotes the average value of the fitness function for the i-th generation. The brainstorm is on for the next y generations, and then it switches to the turning off mode, which runs for the next z generations. When the brainstorm is turned off, it cannot be turned on again for the next b generations even if the "turn on" condition is fulfilled. The n, x, y, z and b values are defined by the user. If the brainstorm is turned on, it modifies the parents' pick up method—the actual average population fitness is added to every individual's fitness value. The brainstorm "turning off" mode linearly decreases the brainstorm modifier. The following function defines the value of the brainstorm modifier $BN(i)$ for a generation i:

$$BN(i) = \begin{cases} 0 & \text{if turned off,} \\ & i < t, \\ F_{\text{avg}}(i) & \text{if turned on,} \\ & t \le i < t+y, \\ \dfrac{(t+y+z)-i}{z} F_{\text{avg}}(i) & \text{if turning off,} \\ & t+y \le i < t+y+z, \end{cases}$$

where t is number of the generation for which the brainstorm is turned on.

The way the brainstorm affects the population average fitness and the best found individual fitness is clearly shown in Fig. 5. We can see that, starting from the 100-th generation, the brainstorm is on. The average fitness function decreases, but the best value of the fitness function increases more rapidly than that for previous generations. After y generations, the brainstorm factor starts to linearly decay during z generations.

The brainstorm operator (population state) is a remedy for a typical situation when the population starts to concentrate only around one local maximum area. In such a population, all individuals are very "close" to each other and so are their fitness values. That is why the whole average fitness starts to become constant. In the presented algorithm, this situation triggers off the brainstorm, which helps the search to escape from this situation (so the population is diverse again). As we can see from Fig. 5, even though the average population fitness has dropped down,

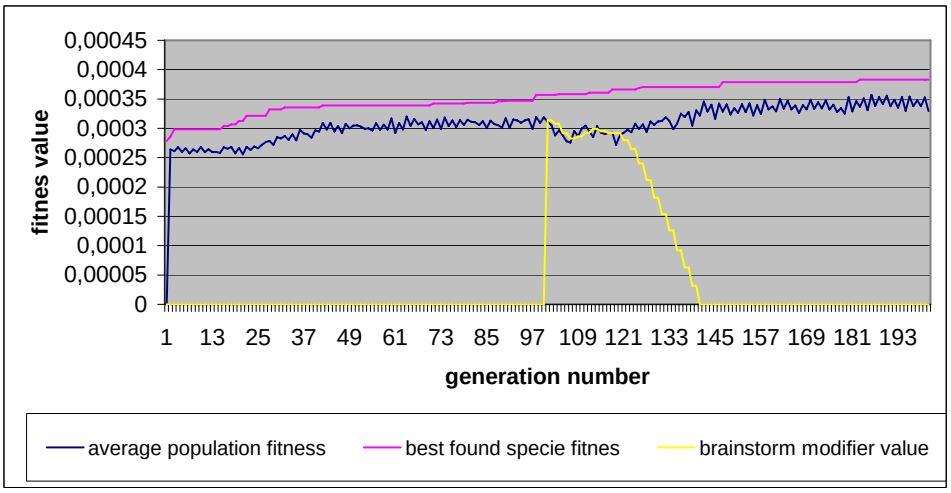

Fig. 5. Brainstorm fitness modifier and an average population fitness.

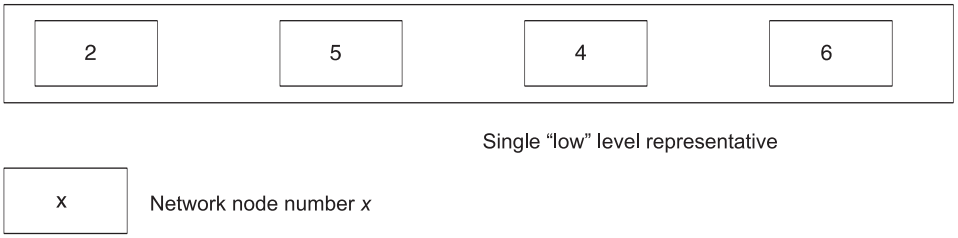

Fig. 6. Low level individual representation.

the fitness of the best individual ever found increases during the brainstorm quicker than before it.

4.3. "Low" Level of the Algorithm. The "low" level is based on a hierarchical algorithm approach (Radcliffe and Surry, 1994), even though its existence does not actually make the presented algorithm hierarchical, nor a typical EA at all. For example, on the one hand, we have two levels, which indicates a hierarchical algorithm, but on the other hand, there is no fitness function defined for the "low" level and there is no typical population, either. These are the reasons why the algorithm presented in this work can rightly be called "quasi-hierarchical".

At the "low" level of the algorithm, each chromosome is represented by a list of network node identifiers that a route goes through. Node identifiers are individuals' genes, which means that the number of genes for each individual can be different and that its neighbors determine the value of a gene. For instance, Chromosome 2546 denotes a route originating in Node 2, going through Nodes 4 and 5, and finally reaching Node 6. An example of a "low" level representation is shown in Fig. 6.

There is no regular fitness function defined for a "low" level, because we cannot evaluate individual routes, and we can only evaluate the whole set of routes. The selection procedure is based on a simple idea which is best described by the following sentence: "If we cannot directly tell which routes are somehow better than others, we have to believe that we will probably find better routes in better fit sets of them (better "high" level individuals)." In order to select two "low" level parents, we select a "high" level one and randomly pick up one of his genes, which is in fact a "low" level individual. Next, we select another "high" level individual and take one of its genes that represents the same commodity as the first already chosen "low" level parent. In the crossing operator, the "high" level individuals from which the "low" level parents come are reintroduced into the new "high" level population. If a chance decides that the "low" level crossing operator is not to be used, both "high" level individuals stay the same like they did before. If the "low" level crossing operator is used, they will contain a modified version of the routes that were randomly chosen to be the "low" level parents. The selection process of "low" level parents is shown in Fig. 7.

The two selected "low" level individuals of the same commodity (two routes with the same source and destination nodes) are crossed by random partition of both individuals into two pieces. The first part from the first parent is then joined with the second part of the second parent, which produces the first offspring, and the second off-

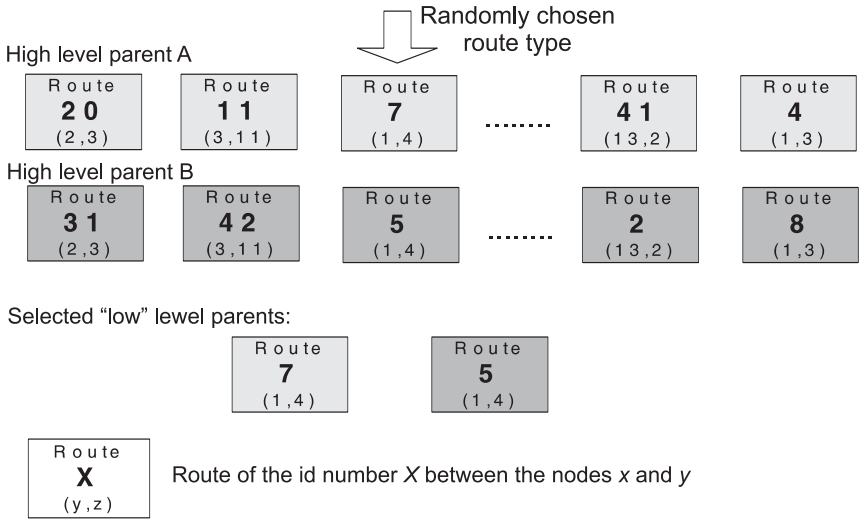

Fig. 7. "Low" level parent selection.

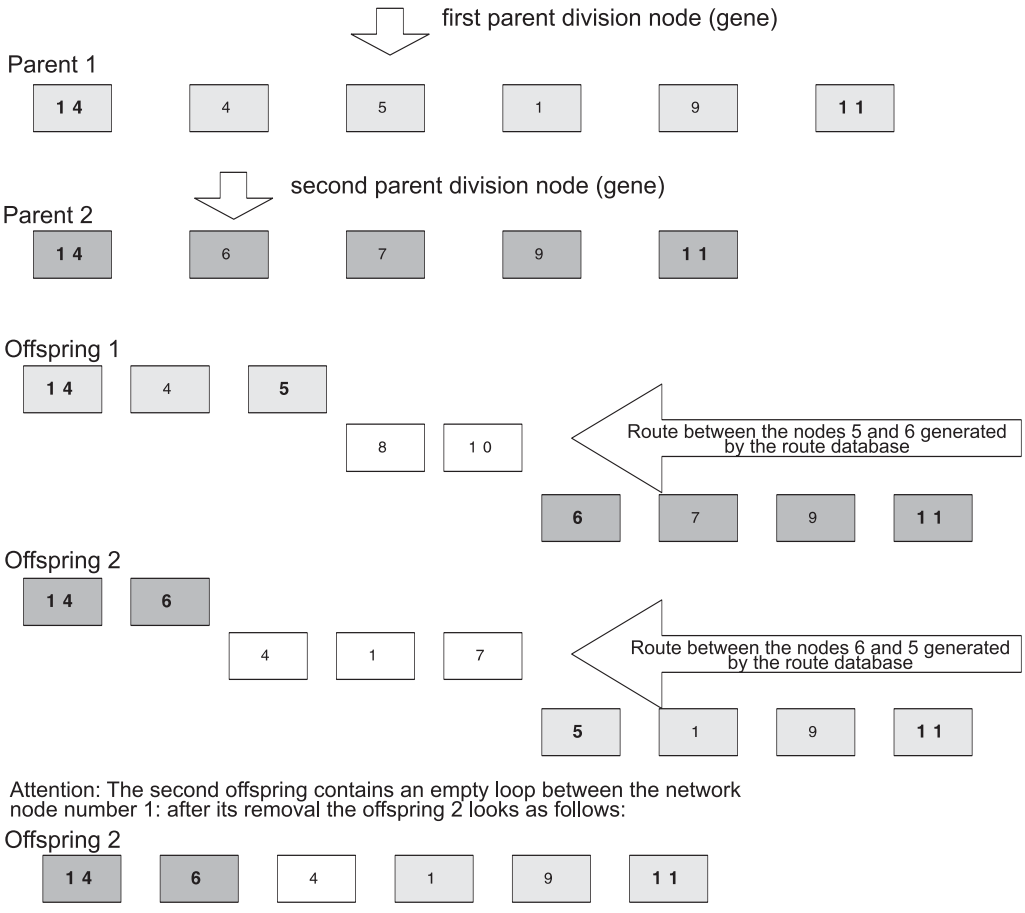

Fig. 8. "Low" level crossing operation.

spring is constructed analogously. The two pieces do not have to fit each other (in fact, they will not in most cases). Therefore, we join them using a randomly generated route between the joint nodes from the database of routes. The "low level" crossing operation is shown in Fig. 8.

The "low" level mutation operator first checks the mutation chance for every "high" level individual. If the "low" mutation operator is to be used, we randomly pick up a "high" level individual gene, which is a "low" level individual, and mutate it by replacing a randomly chosen

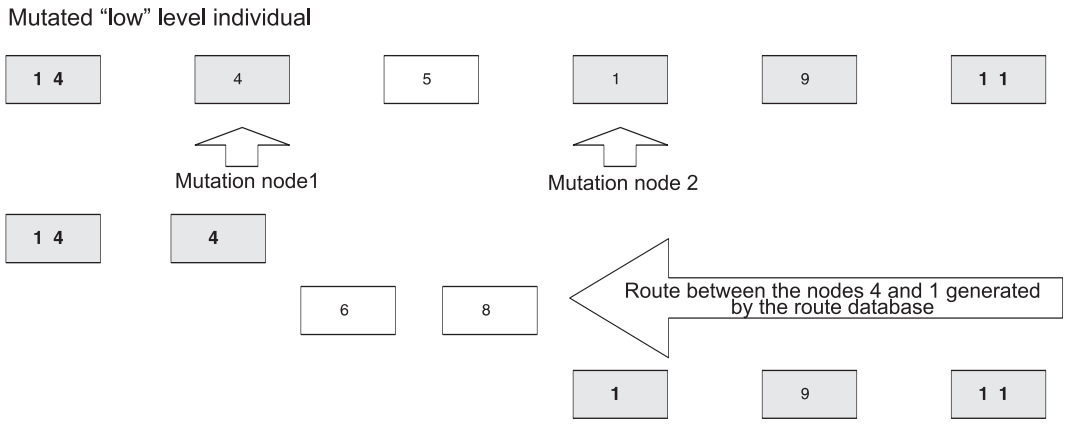

Fig. 9. "Low" level individual mutation.

part of it by a proper (having the same start and end node identifiers as the replaced part) route proposition from the database of routes. It should be noted that the "low" level crossover and mutation operators might produce a route that contains empty loops, which are removed from it before it is shown to the database of routes as a new route proposition. The "low" level mutation operation is also shown in Fig. 9.

4.4. General Algorithm Overview. The general algorithm overview is displayed in Fig. 10. We show there the order of algorithm stages and the role of the database of routes.

5. Results

The algorithm was coded in C++. For tests we applied the same networks as in (Walkowiak, 2004a; 2004b), consisting of 10, 14, 18 and 36 nodes and having various numbers of links. Table 1 briefly summarizes the parameters of all 36-node networks. The first column specifies parameter names, and the next columns include the values of these parameters for each network. Let the bandwidth unit (BU) denote an arbitrary unit of the bandwidth, e.g., 1 Mb/s. We assume that, for all networks, the capacity of each link is 5000 BU. Since, according to the theoretical analysis presented in (Walkowiak, 2004a; 2004b), the LFN function of a lost flow depends on the node degree, in numerical experiments we chose networks with various values of the average node degrees. It is assumed that there is a requirement to set up a connection for each direction of every node pair. Thus, the total number of demands (commodities) is $n(n-1)$. Each demand is defined by a source node, a destination node and a flow requirement. Several demand patterns are examined for each network. However, all demands are homogenous and the flow requirement is the same for each demand. For instance, for Network 104 we performed 17 simulations starting with a flow requirement of each demand equal to 48 BU, the largest value of the flow requirement being 64 BU.

Table 1. Parameters of the tested networks.

Name of network	104	114	128	144	162
Number of nodes	36	36	36	36	36
Number of links	104	114	128	144	162
Topology	irregular mesh				
Node degree (average)	2.88	3.17	3.56	4.00	4.50
Number of tests	17	8	9	9	7

The first step of numerical experiments was tuning the algorithm. Parameter settings used for further simulations were selected after a high number of simulation runs. Configuration selection was divided into two phases. In the first phase, the best possible values of different parameters (the probabilities of "high" and "low" level crossing and mutation, the probability of one-point/uniform crossover and the "brainstorm" parameter set) were selected (this selection was done arbitrarily), and the algorithm was to run for all the possible combinations of them. This way of setting tests seemed to be quite time-consuming (there were 32 parameter combinations to go through), and thus all these tests were made for a small number of individuals in the population (400) and a low number of generations (100), which were fixed arbitrarily. In the second phase, the best parameter settings were used for computation but with much higher values of individuals and numbers of generations (1200 individuals and 1000 generations for 10-node networks; 2000 individuals and 1000 generations for 14-node networks; 4000 and 1200 for 18-node networks; 2000 and 1000 for 36-nodes networks). Typical times of algorithm runs on a machine equipped with a 1.3 GHz processor were about 50 minutes. The algorithm was initiated with a set of feasible routes. The database contained all possible routes whose lengths did not exceed 4.

The results of the initial phase are shown in Table 2. This table contains the average percentage of the best re-

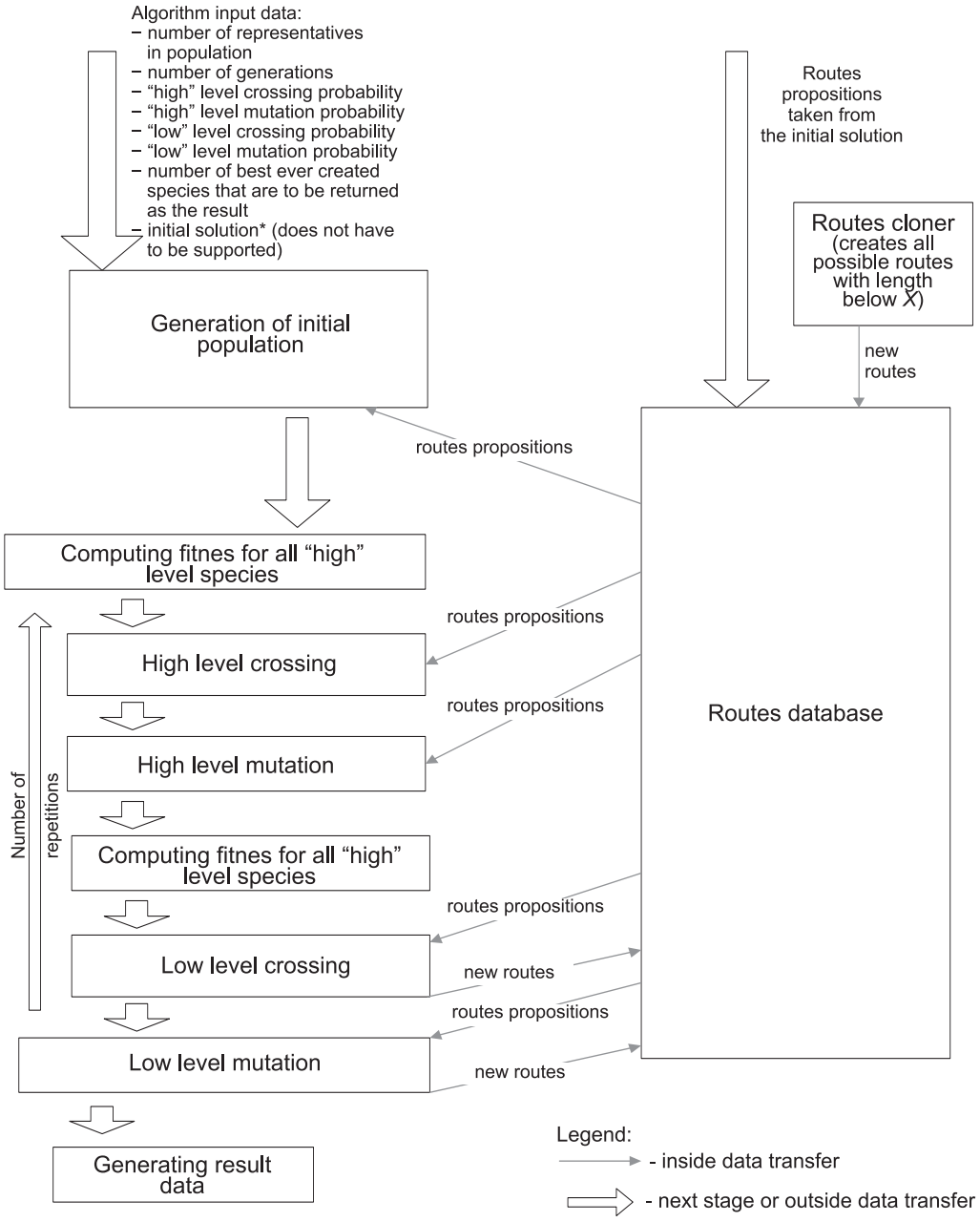

Fig. 10. General algorithm overview.

sults found for a specified network type, which was computed as follows: we compared solutions for a specified network produced by all settings such that the best result found was assigned a value of 1 (100%) and the others – proportionally lower values. The average of all these values for a specified network type (e.g., a 14-node one) for each configuration is the average percentage of the best result found for a specified configuration.

The content of Table 2 can be better represented as a function of configuration where the function value is the average percentage of the best solution. In Fig. 11 we can clearly see that the shape and values of the presented functions are different depending on the network type. For 10-node networks, all algorithm settings were equally effective. For 14-node networks, still all settings were equally effective, but their effectiveness descreased. For 18-node networks, the effectiveness was on the same level as for 14-node networks, but we can see there are three configurations considerably better than the others. For 36-node networks, the effectiveness continuously decreased and the diversity between the effectiveness of different settings became even bigger. Finally, a setting considered the best in our opinion was picked for further research. This is shown in Table 3.

Table 2. Results of the initial phase.

Configuration		Normalized average percentage of the best found solution			
Nr.	$X/Y/Z$	10-node networks	14-node networks	18-node networks	36-node networks
1	0/0/0	94.16%	69.79%	72.61%	45.68%
2	0/0/1	88.20%	88.66%	70.68%	54.51%
3	0/1/0	93.50%	92.56%	73.39%	66.86%
4	0/1/1	91.35%	77.45%	76.75%	54.82%
5	0/2/0	91.80%	79.76%	76.03%	66.29%
6	0/2/1	91.36%	79.79%	81.31%	66.32%
7	0/3/0	94.32%	94.72%	71.16%	72.34%
8	0/3/1	96.08%	75.03%	75.85%	84.55%
9	1/0/0	97.27%	82.53%	77.93%	59.69%
10	1/0/1	95.74%	83.85%	72.71%	46.35%
11	1/1/0	96.87%	97.66%	72.30%	78.47%
12	1/1/1	98.39%	83.07%	75.09%	70.55%
13	1/2/0	96.97%	93.36%	74.20%	72.74%
14	1/2/1	100.00%	97.16%	75.45%	91.42%
15	1/3/0	93.83%	91.05%	76.67%	72.36%
16	1/3/1	94.92%	93.35%	99.59%	76.87%
17	2/0/0	96.60%	94.87%	74.66%	88.13%
18	2/0/1	99.82%	98.73%	75.96%	58.02%
19	2/1/0	97.51%	87.99%	97.07%	85.18%
20	2/1/1	99.69%	89.14%	78.01%	100.00%
21	2/2/0	95.11%	94.03%	100.00%	78.70%
22	2/2/1	97.90%	96.17%	78.38%	75.49%
23	2/3/0	98.37%	96.78%	76.98%	90.98%
24	2/3/1	96.30%	90.59%	83.61%	85.51%
25	3/0/0	98.58%	82.56%	77.56%	81.51%
26	3/0/1	96.63%	76.33%	74.95%	71.45%
27	3/1/0	95.64%	94.11%	78.67%	62.44%
28	3/1/1	97.64%	94.70%	71.24%	53.96%
29	3/2/0	96.51%	79.28%	67.38%	62.36%
30	3/2/1	97.02%	83.00%	72.39%	57.70%
31	3/3/0	98.63%	81.81%	70.69%	63.25%
32	3/3/1	96.15%	100.00%	77.32%	56.46%

Setting $X/Y/Z$

X: hcros/hmut/lcros/lmut	Y: uniform crossover possibility	Z: brainstorm
0 – 0.5/0.1/0/0	0 – 0	0 – turned off
1 – 0.5/0.1/0.3/0.2	1 – 0.5	1 – turned on
2 – 0.5/0.1/0.6/0.4	2 – 0.75	
3 – 0/0/0.7/0.5	3 – 1.0	

Table 3. Best configurations.

Best configurations	14 (1/2/1)
High level crossing	0.5
High level mutation	0.1
Low level crossing	0.3
Low level mutation	0.2
Uniform crossover possibility	0.75
Brainstorm	On

The problems addressed in this work are typical GA-hard problems, where the space of feasible solutions constitutes an extremely small part of the space of all possible solutions. A very common problem in such cases for GA-based algorithms is fast convergence to suboptimal solutions. This concentration may be easily seen in Figs. 12–14 as for a long period the population fitness and the best ever found individual fitness remain on the same level. Our algorithm tries to break through this stagnation by using the brainstorm described above. All individuals are rewarded in accordance with their fitness values when the algorithm picks up parents for crossing. This enables the population become more diverse, which can be clearly seen in figures as the temporary average population fitness deteriorates. However, the average population fitness decreases and the brainstorm period enables the algorithm to find better individuals during the brainstorm period or immediately after. This is also confirmed by the results when the same settings using the brainstorm produced better results than the same settings without it. Certainly, the brainstorm does not always provide very good results. For instance, in Fig. 12 we can see that the brainstorm improves the solution only after ten attempts. The very model of the algorithm run is presented in Fig. 13. The brainstorm clearly helps the algorithm to find better solutions.

It is difficult to estimate the quality of the HEFAN because, as has been mentioned above, the only way to find an optimal solution for 0–1 problems is the branch-and-bound method, which can be used only for very small networks (about 10 nodes). One of the authors developed a branch-and-bound algorithm for a restricted version of the optimization problem considered (Walkowiak, 2004a). Also, a genetic algorithm was employed for the same problem (Walkowiak, 2001). However, genetic and B&B algorithms can process only a limited set of candidate routes (not all possible routes are taken into account). Summarizing all 309 tests made for 8 networks with the number of nodes varying from 10 to 14, the genetic algorithm gave results only by 0.7% worse than the optimal ones. The evolutionary algorithm proposed in this work includes important extensions compared with that pro-

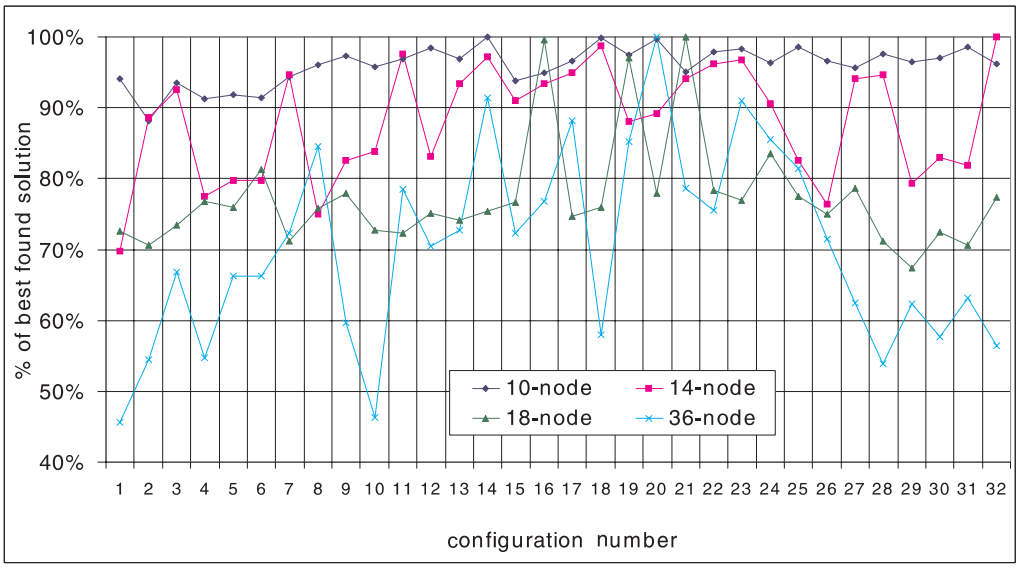

Fig. 11. Average percentage of the best found solution as a function of the configuration.

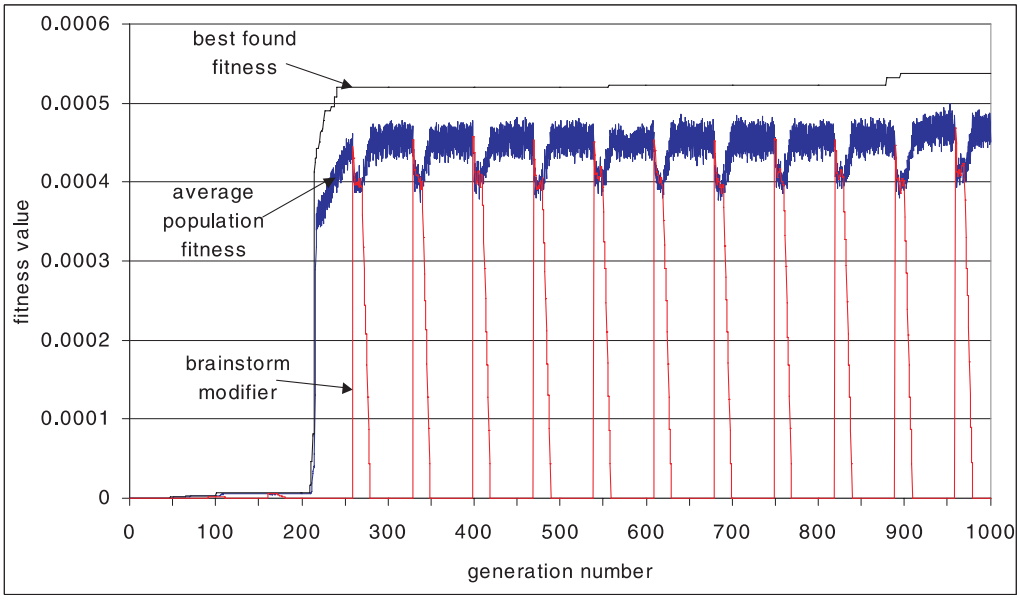

Fig. 12. Algorithm run for the configuration 1/2/1, 10-node and 38-arc network.

posed in (Walkowiak, 2001). The most significant modification is the low level that enables searching the whole solution space. Also, new operators are added. Concluding, we expect that the evolutionary algorithm – HEFAN – presented above is able to find results close to optimal ones.

For reference, we apply a heuristic algorithm FDP based on the Flow Deviation (FD) method proposed in (Walkowiak, 2003; 2004b) for restricted and unrestricted versions of the PRA problem. Furthermore, we also present the results of a newly developed heuristic called LRH that combines FDP and Lagrangian relaxation techniques (Walkowiak, 2006). We focus on the most interesting cases, i.e., the largest networks consisting of 36 nodes presented in Table 1. We present the results of the HEFAN using the combination of parameters given in Table 3.

To compare the results, we apply the *competitive ratio* performance indicator. The competitive ratio, which indicates how well an algorithm performs for a particular network and a demand pattern, is defined as the difference between the result obtained for the analysed algorithm and the minimum value of LFN yielded by the best algorithm. For instance, if FDP yields 2500, LRH yields 2200 and the HEFAN yields 2000 for the same simulation

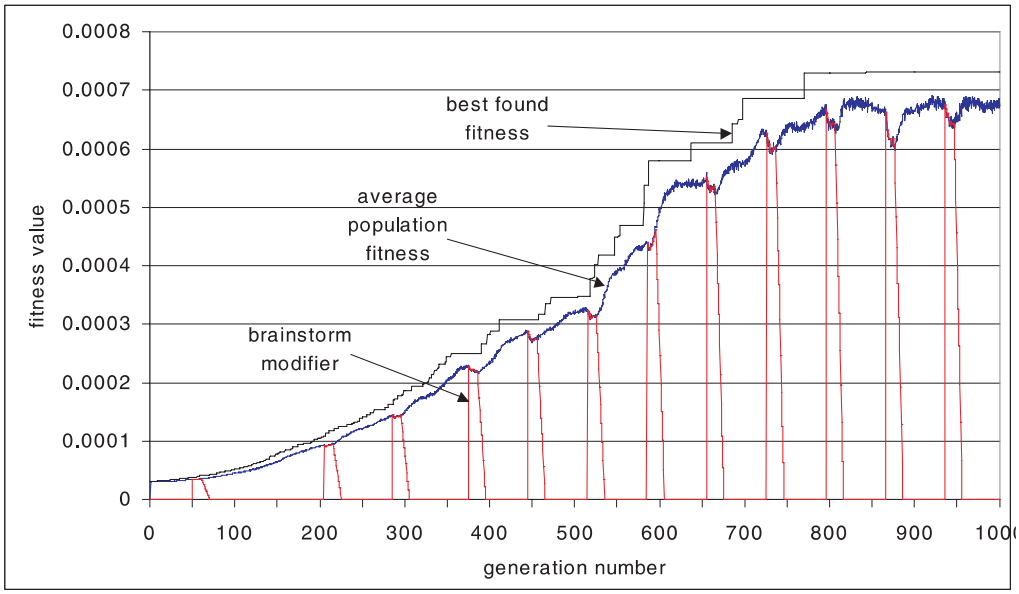

Fig. 13. Algorithm run for the configuration 1/2/1, 36-node and 128-arc network.

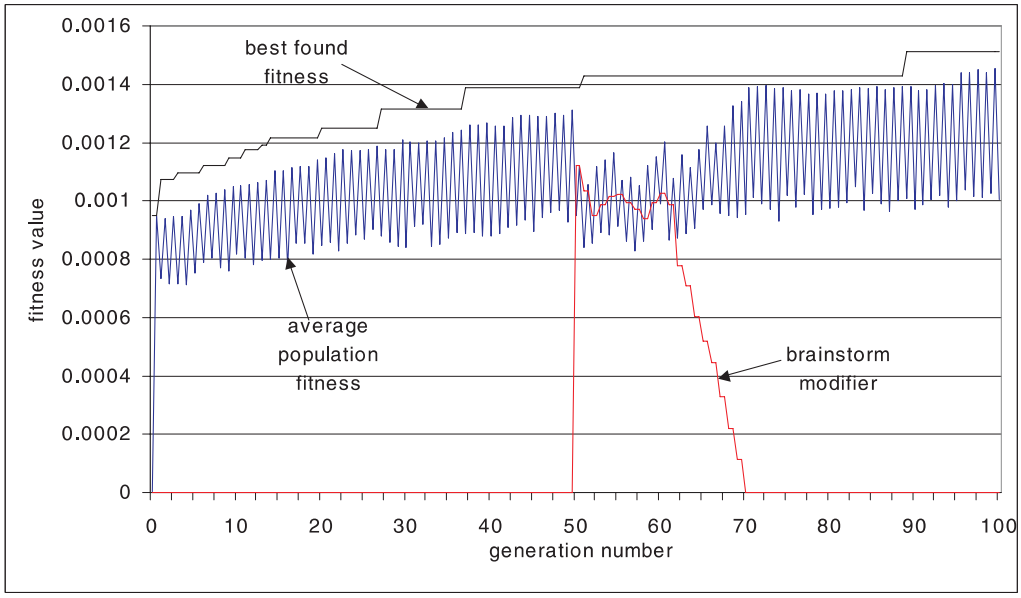

Fig. 14. Algorithm run for the configuration 3/2/1, 10-node and 42-arc network.

case; the competitive ratio of FDP is calculated as follows: $(2500 - 2000)/2500 = 20\%$. The competitive ratio indicates the quality of a particular algorithm compared with the best found result. A low value of the competitive ratio means that the algorithm finds a solution close to the best results obtained in a given test. Notice that the competitive ratio must be in the range $[0, 100\%]$. For the presentation of aggregate results, we apply the *average competitive ratio*, which is the average value of competitive ratios for a given network topology.

In Table 4 we report the average competitive ratios of the tested algorithms obtained for various networks. Table 5 shows the ranking of all tested algorithms. For instance, the fourth column of Table 5 demonstrates that the HEFAN produces the best results in 1 out of 17 simulation runs for Network 104, 5 of 8 simulation runs for Network 114, etc. In almost all of the tested and presented cases, the HEFAN outperforms the FDP algorithm. Only for low-connected Network 104 FDP offers better results. The performances of LRH and the HEFAN are compara-

ble, yet they depend on the network topology. Both algorithms try to search as many candidate routes as possible. However, the number of candidate routes depends on network connectivity.

Table 4. Average competitive ratio of FDP, LRH and the HEFAN for various networks.

Network	FDP	LRH	HEFAN
104	7.90%	0.00%	20.58%
114	20.67%	15.29%	1.83%
128	24.36%	13.39%	11.21%
144	24.54%	3.17%	17.71%
162	22.97%	0.00%	16.22%

Table 5. Ranking of FDP, LRH and the HEFAN for various networks.

Network	FDP	LRH	HEFAN
104	3	17	1
114	1	4	5
128	1	2	6
144	1	5	5
162	2	7	2

The results collected during the search for the best routes show that all of the "low" level parameters are necessary to find the best route set. This feature is even common for small networks (10 nodes), where the "low" level is not needed for finding new route proposals (for the fitness function used we have enough routes in the database), but the algorithm seems to find better route proposals much faster than when it does not use "low" level operators at all. The most impressive example of how useful the "low" level can be regards large networks. For such networks, there are node pairs which demand long route proposals (of lengths over 8 hops). Initializing the database of routes with all such proposals (and longer ones because some of them may also be useful) would certainly make the algorithm unable to work in a reasonable time. On the other hand, we have the presented "low" level operator, which seem to be a good answer to this problem. However, the tests should be done for populations bigger than 2000 and generation numbers surpassing 1000, because some of the results are quite disappointing.

6. Concluding Remarks and Further Work

The main idea of the presented algorithm based on a two-level structure and the proposed performance of the "low" level seems to be a good direction for future work and improvements. The results reported in this paper show that there is still an improvement potential in modifying parents' selection and finding better "brainstorm" mode variables (it looks like the algorithm has big problems escaping from local optima). Another improvement that was not mentioned yet regards finding a way to decrease the number of node pairs that have to be computed within the same time period, since problem complexity increases quadratically with the increase in the network size (i.e., the number of nodes). Very optimistic seems the possibility of making the "low" level of the proposed algorithm more independent of the "high" level, especially by adding a regular "low" level fitness function (first attempts suggest that significant improvements can be obtained). Another step ahead may be to expand the current algorithm so that it can also design the network topology for given demands and route sets. It should be underlined that the proposed EA can be applied to a wide range of optimization problems encountered in c-o networks using as objective functions the network cost, delay and many other criteria. The algorithm is not limited only to the function LFN.

We also consider the possibilities of extending the algorithm by the creation of individuals' classes based on a proper definition of the distance between two representatives. This, we believe, may be a much better panacea for premature convergence than the "brainstorm". Even better perspectives can be offered by applying a messy-GA algorithm structure, since it is possible that some individuals might be significantly improved by changing only a small set of genes which are far away from each other in the current chromosome coding. In the messy-GA algorithm structure, the position of a particular gene in the chromosome does not change its meaning. Then, it is possible that genes from such an important set will be grouped close to each other and the probability of a positive change in this set will be highly increased. However, before a messy-GA algorithm structure can be applied to the HEFAN, the problem of computing the fitness function based only on partial information has to be addressed.

References

Corne D., Oates M. and Smith D. (Eds.) (2000): *Telecommunications Optimization: Heuristic and Adaptive Techniques*. — New York: Wiley.

Davis L. (1996): *Handbook of Genetic Algorithm*. — New York: Van Nostrand Reinhold.

Elbaum R. and Sidi M. (1996): *Topological design of local area networks using genetic algorithms*. — IEEE/ATM Trans. Networking, Vol. 4, No. 5, pp. 766–778.

Grover W. (2004): *Mesh-Based Survivable Networks: Options and Strategies for Optical, MPLS, SONET and ATM Networking*. — Upper Saddle River, NJ: Prentice Hall.

Fratta L., Gerla M. and Kleinrock L. (1973): *The flow deviation method: An approach to store-and-forward communication network design*. — Networks, Vol. 3, No. 2, pp. 97–133.

Jæger B. and Tipper D. (2003): *Prioritized traffic restoration in connection oriented QoS based networks*. — Comput. Commun., Vol. 26, No. 18, pp. 2025–2036.

Karp R. (1975): *On the computational complexity of combinatorical problems*. — Networks, Vol. 5, No. 1, pp. 45–68.

Kasprzak A. (2003): *Exact and approximate algorithms for topological design of wide area networks with non-simultaneous single commodity flows*. — Lect. Notes Comput. Sci., Vol. 2660, pp. 799–808.

Kwaśnicka H. (1998): *Genetic and Evolutionary Algorithms—an Overview*. — Wrocław: University of Technology Press.

Michalewicz Z. (1996): *Genetic Algorithms + Data Structures = Evolution Programs, 3-rd Ed*. — Berlin: Springer.

Murakami K. and Kim H. (1996): *Virtual path routing for survivable ATM networks*. — IEEE/ATM Trans. Networking, Vol. 4, No. 2, pp. 22–39.

Nilsson P., Pióro M. and Dziong Z. (2003): *Link protection within an existing backbone network*. — Proc. Int. *Network Optimization Conf., INOC*, Evry, Paris, pp. 435–440.

Pióro M. and Medhi D. (2004): *Routing, Flow, and Capacity Design in Communication and Computer Networks*. — San Francisco: Morgan Kaufman.

Przewoźniczek M. (2003): *Genetic algorithms in use of routes finding in computer connection oriented networks*. — M.Sc. thesis, Wrocław University of Technology, Wrocław, Poland.

Przewoźniczek M. and Walkowiak K. (2005): *Evolutionary algorithm for congestion problem in connection-oriented networks*. — Lect. Notes Comput. Sci., Vol. 3483, pp. 802–811.

Radcliffe N. and Surry P. (1994): *Co-operation through hierarchical competition in genetic data mining*. — Tech. Rep., No. EPCC-TR94-09, Edinburgh Parallel Computing Center.

Riedl A. (1998): *A versatile genetic algorithm for network planning*. — Proc. 4-th *Open European Summer School, EUNICE'98*, Munich, Germany, pp. 97–103.

Riedl A. (2002): *A hybrid genetic algorithm for routing optimization in IP networks utilizing bandwidth and delay metrics*. — Proc. IEEE Workshop *IP Operations and Management, IPOM,* Dallas, pp. 166–170.

Walkowiak K. (2001): *Genetic approach to virtual paths assignment in survivable ATM networks*. — Proc. 7-th Int. Conf. *Soft Computing MENDEL*, Brno, Czech Republic, pp. 13-18.

Walkowiak K. (2003): *A new approach to survivability of connection oriented networks*. — Lect. Notes Comput. Sci., Vol. 2657, pp. 501–510.

Walkowiak K. (2004a): *A branch and bound algorithm for primary routes assignment in survivable connection oriented networks*. — Comput. Optim. Applic., Vol. 27, No. 2, pp. 149–171.

Walkowiak K. (2004b): *A new method of primary routes selection for local restoration*. — Lect. Notes Comput. Sci., Vol. 3042, pp. 1024–1035.

Walkowiak K. (2004c): *Survivable online routing for MPLS traffic engineering*. — Lect. Notes Comput. Sci., Vol. 3266, pp. 288–297.

Walkowiak K. (2006): *Lagrangean heuristic for primary routes assignment in survivable connection-oriented networks*. — Tech. Rep., Wrocław University of Technology, Wrocław.

White A.R.P., Mann J.W. and Smith G.D. (1999): *Genetic algorithms and network ring design*. — Annals Oper. Res., Vol. 86, No. 1, pp. 347–371.

Received: 1 December 2005
Revised: 10 August 2006