
ELSEVIER

Performance Evaluation 48 (2002) 201–223

PERFORMANCE EVALUATION
An International Journal

www.elsevier.com/locate/peva

On open shortest path first related network optimisation problems

M. Pióro [a,b,*], Á. Szentesi [c], J. Harmatos [c], A. Jüttner [c], P. Gajowniczek [b], S. Kozdrowski [b]

[a] *Department of Communication Systems, Lund Institute of Technology, Lund, Sweden*
[b] *Institute of Telecommunications, Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warszawa, Poland*
[c] *Ericsson Research, Traffic Analysis and Network Performance Laboratory, Budapest, Hungary*

Abstract

The paper deals with flow allocation problems in IP networks using open shortest path first (OSPF) routing. Its main purpose is to discuss and propose methods for finding settlements of OSPF link weight system realising the assumed demand pattern for the given network resources (links capacities). Such settlements can result in a significantly better network performance, as compared with the simplified weight setting heuristics typically used nowadays. Although the configuration of the link weight system is primarily done in the network planning phase, still additional re-optimisations are feasible, and in fact essential, in order to cope with major changes in traffic conditions and with major resources' failures and rearrangements.

The paper formulates a relevant OSPF routing optimisation problem, proves its NP-completeness, and discusses possible heuristic approaches and related optimisation methods for solving it. Two basic approaches are considered (the direct approach and the two-phase approach) and the resulting optimisation algorithms are presented. The considerations are illustrated with numerical results.

Keywords: OSPF; Traffic engineering; Network optimisation; Multicommodity flows

1. Introduction

The open shortest path first (OSPF) packet routing protocol [1] is one of the most commonly used *interior gateway protocols* in today's IP networks. OSPF uses shortest paths for routing the packets, applying the equal-cost multipath (ECMP) principle to cope with multiple shortest paths. The packet routing mechanism is therefore relatively simple, and can essentially be summarised as follows: all the packets arriving at an intermediate node t and destined for node u are directed to the next hop along the shortest path from t to u, regardless of the packets' originating nodes. If there are several links outgoing from node t and belonging to the shortest paths from t to u, then the traffic is distributed evenly among these links. The shortest paths to destinations are identified at the network nodes on the basis of the current links weight (metric) system w: each link e is assigned a positive number w_e (weight) and, as a result of

* Corresponding author. Present address: Institute of Telecommunications, Warsaw University of Technology, Nowowiejska 15/19, 00665 Warszawa, Poland. Tel.: +48-22-825-98-20; fax: +48-22-660-7564.
E-mail address: mpp@tele.pw.edu.pl (M. Pióro).

0166-5316/02/$ – see front matter
PII: S0166-5316(02)00036-6

the OSPF link-state flooding mechanism, all the nodes are aware of the weights $w = (w_1, w_2, \ldots, w_E)$ of all network's links.

Note that once the weight system is fixed, it strictly determines all the shortest paths and, consequently, the points (nodes) where packet traffic flows are split to their destinations according to the ECMP rule, not allowing for any flexibility during the network operation. On the other hand, the weight system w can be adjusted and made fit to the current network state, i.e. by assigning infinite weights to the failed links. Here optimisation problems of how the weight system should be calculated and adjusted in order to optimise network performance arise; the investigations in this area have started only recently [2–4].

In the paper we address the capacitated (link capacities are given and fixed) allocation problems related to the OSPF routing. The main theoretical result of the paper, presented in Section 2, is a proof showing that the OSPF flow allocation problem considered in the paper is NP-complete (for the notion of NP-completeness see [5]). Because of the NP-completeness of the OSPF flow allocation problem, one is forced to use heuristic methods for solving them. In Section 3 we discuss heuristic algorithms based on the search in the weight systems space using *local search, simulated annealing* and *Lagrangean relaxation*, and the *branch-and-bound approach*. Next, in Section 4, we discuss a two-phase approach in which the demands are first allocated to single paths (Phase 1) and then an attempt is made (Phase 2) to find a corresponding weight system w generating the set of single demand allocation paths obtained in Phase 1 (this means that the paths found in Phase 1 must be the unique shortest paths with respect to the weight system w). For Phase 1 we consider *mixed linear-integer programming* (MIP), and certain heuristic algorithms based on the *evolutionary algorithms and simulated allocation* meta-heuristics. For Phase 2 we discuss known and new linear programming formulations for testing the shortest paths uniqueness, and for finding, when they exists, the weight systems that generate the path sets obtained in Phase 1. We illustrate the effectiveness of the proposed algorithms with numerical examples (Section 5) and draw some conclusions (Section 6).

2. Basic problem and its NP-completeness

The main question dealt with in this section is the existence of a feasible OSPF link weight system for given demands matrix and links capacities. Hence, we ask whether there exists a system of weights that generates flows realising the demands such that the resulting links loads do not exceed the given links capacities. We investigate the computational complexity of the problem.

2.1. Problem formulation

We consider the following OSPF flow allocation problem, abbreviated with FAP:

- indices:
  - $d = 1, 2, \ldots, D$ demands
  - $j = 0, 1, \ldots, m(d)$ path for flows realising demand d
  - $e = 1, 2, \ldots, E$ links
- constants:
  - h_d volume of demand d to be realised
  - a_{edj} 1 if link e belongs to path j realising demand d, 0 otherwise
  - y_e capacity of link e

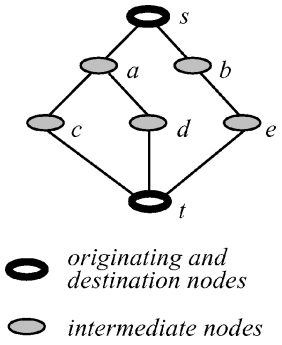


Fig. 1. OSPF flow splitting rule.

- variables:
  w_e weight of link e (non-negative continuous variable), $w = (w_1, w_2, \ldots, w_E)$
- constraints:

$$\sum_j x_{dj}(w) = h_d, \quad d = 1, 2, \ldots, D, \tag{2.1}$$

$$\sum_d \sum_j a_{edj} x_{dj}(w) \leq y_e, \quad e = 1, 2, \ldots, E. \tag{2.2}$$

Above, $x_{dj}(w)$ denotes the flow realising demand d on path j, implied by the links weight system w. For a given system w, the flows $x_{dj}(w)$ are computed according to the ECMP rule. The rule is illustrated in Fig. 1: the shortest paths *s–a–c–t* and *s–a–d–t* realise 1/4 of the total demand volume between nodes s and t, whilst the shortest path *s–b–e–t* realises the remaining 1/2 of the volume. Note that the flow functions $x_{dj}(w)$ are not given explicitly, so FAP is not a mathematical programme. In order to make the ECMP flow splitting procedure consistent, OSPF usually assumes that the link weights are positive, so there are no loops in the shortest paths. Constraints (2.1) in FAP guarantee that all demands are realised, and constraints (2.2)—that links loads do not exceed their capacities. The weight systems space in FAP can be further limited; for instance, two simple weight spaces assuring the consistency of the weight systems are defined by

$$w_e \in \{1, 2, \ldots, K\}, \quad e = 1, 2, \ldots, E \text{ (integer systems)}, \tag{2.3}$$

$$w_e \in [1, K], \quad e = 1, 2, \ldots, E \text{ (continuous systems)}. \tag{2.4}$$

2.2. NP-completeness result

Now we shall demonstrate that FAP is an NP-complete problem. To do this we shall show that the so-called X3C problem (exact cover by 3-sets) ([SP2], p. 221 in [5]) can be transformed to (a simplified version of) FAP. X3C is known to be NP-complete and is as follows:

1. *instance*: set $X = \{x_1, x_2, \ldots, x_p\}$ of $p = 3q$ elements ($|X| = p$) and a family C of n 3-subsets of X ($C = \{C_1, C_2, \ldots, C_n\}$, $C_i \subseteq X$, $|C_i| = 3$, $i = 1, 2, \ldots, n$), $n \geq q$.
2. *question*: does C contain a subfamily $C' \subseteq C$, of q ($|C'| = q$) pair-wise disjoint subsets of X?

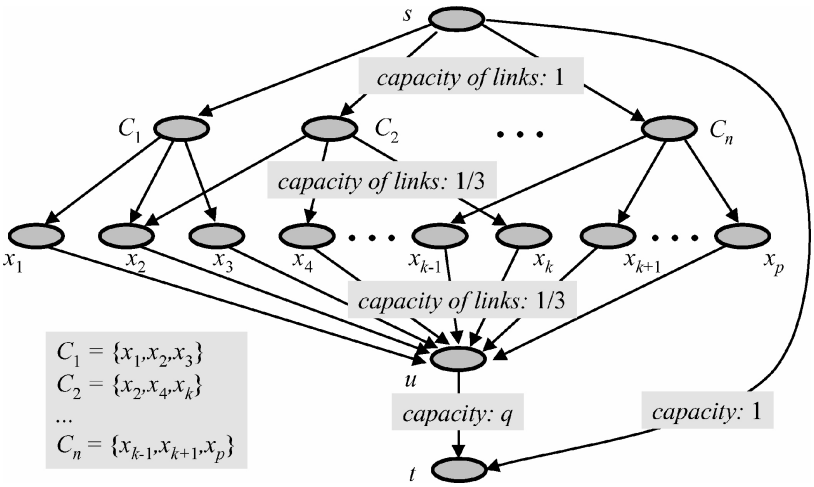


Fig. 2. The single-commodity flow graph.

Consider the directed single-commodity flow graph depicted in Fig. 2. Vertex s is the source, vertex t is the sink. The vertices in the upper row correspond to the 3-sets of family C, whilst the vertices in the lower row correspond to the elements of set X. The edges between the two rows reflect the incidence relation between family C and set X: vertex C_i is connected to vertex x_k if, and only if, $x_k \in C_i$ (in the considered example $C_1 = \{x_1, x_2, x_3\}$, $C_2 = \{x_2, x_4, x_k\}$ and $C_n = \{x_{k-1}, x_{k+1}, x_p\}$). In the sequel we shall assume that $X \subseteq \cup C$, i.e. that family C covers set X (otherwise X3C is trivial).

It is easy to see that for the particular edges capacity values assumed in the considered graph, the value of the maximal flow from s to t is equal to $q + 1$. To see this we first note that the flow cannot be greater than $q + 1$, since no more flow than $q + 1$ can be received by the sink. On the other hand, we can saturate all the edges incoming to vertex u and hence achieve the maximal flow. To do this, for each element of X we select one 3-subset from C that contains the considered element and assign flow 1/3 to the corresponding edge. This operation determines how much flow must be assigned to each edge from the source to the first row of vertices. The maximal flow from s to t can be easily found in the described way in polynomial time.

Now let us constraint the admissible flows in the considered graph to the so called equal-split flows (ES-flows). A flow f is an ES-flow if for each vertex v the flows assigned to the edges outgoing from v are either equal to 0 or to some fixed, vertex-dependent positive value. In other words, for any fixed vertex v and each edge of the form (v, w), there exists a number $z(v)$ such that $f(v, w) = 0$ or $f(v, w) = z(v)$. The basic observation leading to our NP-completeness result is that the answer to the question in X3C is positive if and only if the maximal ES-flow in the considered graph is equal to $q + 1$.

Suppose the subfamily $C' = \{C_{i(1)}, C_{i(2)}, \ldots, C_{i(q)}\}$ exactly covers set X. We assign flow equal 1 to all edges $(s, C_{i(j)})$ for $j = 1, 2, \ldots, q$ (i.e. $f(s, C_{i(j)}) = 1$, $j = 1, 2, \ldots, q$), and flow equal 0 to the rest of the edges of the form (s, C_l). This assignment will force $f(x_k, u) = 1/3$ for $k = 1, 2, \ldots, p$. Finally, assigning $f(s, t) = 1$ we arrive at an ES-flow with value $q + 1$. Conversely, if the maximal ES-flow is

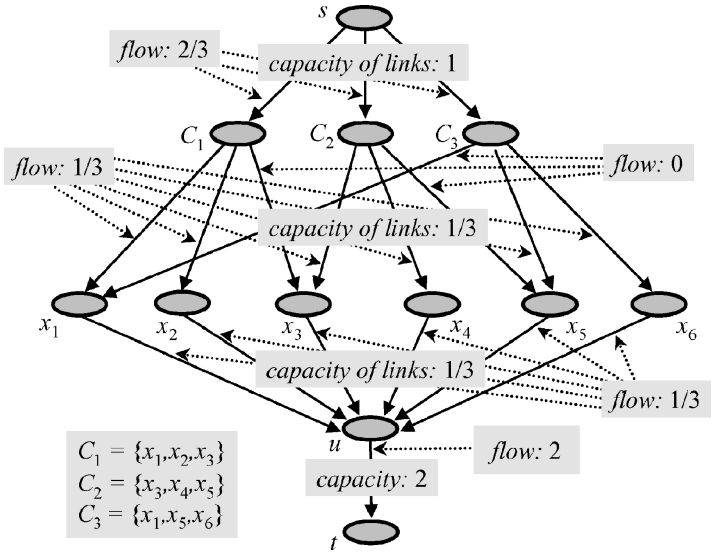


Fig. 3. The single-commodity flow graph.

equal to $q + 1$ then, due to the equal-split assumption, this can be achieved only in one way: flow equal to 1 is assigned to edge (s, t) and to exactly q out of p edges of the form (s, C_l) (with $f(s, t) < 1$ the flow value $q + 1$ could not be achieved because of the capacity q assigned to link (u, t)). Note that this is the reason why edge (s, t) is necessary: without it, it would be possible, as illustrated in Fig. 3, to find an ES-flow with value q even if there would be no exact 3-cover.) The only possible way to maintain flow of value q down in the main part of the graph (i.e. the left part, without edge (s, t)) is to saturate all edges incoming to vertex u. This implies that the vertices C_l with $f(s, C_{i(j)}) = 1$ define the family C' exactly covering set X.

Thus, we have proved our observation: the answer to the X3C question is positive if and only if the maximal ES-flow is equal to $q + 1$. Hence, if we were able to find in polynomial time an ES-flow equal to $q + 1$, or show that such a flow does not exist, then we would answer X3C in polynomial time. This proves that the following ES-flow problem (ESF) is NP-complete:

- *instance*: integers p, q, n such that $p = 3q$ and $n \geq q$, and a graph of the structure depicted in Fig. 1.
- *question*: does there exist an ES-flow of value $q + 1$?

Note that any instance of ESF can be solved by solving a corresponding instance of FAP. Such instances of FAP have only one demand $d = 1$ (between s and t) with volume h_1 equal to $q + 1$, and the path structure and links capacities specified by the graph in Fig. 2. The task is to find a links (edges) weight system that combined with the ECMP rule defines an ES-flow f answering the question in ESF. Of course, every ES-flow in the considered graph can be generated by the weight system defined by assigning weights $w = 1$ to all edges with positive flows in the main part of the graph (these flows are either 1, $\frac{1}{3}$ or q), weight $w = 4$ to edge (s, t), and weights $w = \infty$ to all other edges. Thus any algorithm solving FAP solves also ESF, and hence X3C. This proves that FAP is NP-complete.

2.3. Another formulation

In a well-known paper [2] the authors consider piece-wise convex linear functions $f_e(x)$ for penalising breaking constraints (2.2). The problem is

$$\text{minimise } C = \sum_e f_e \left(\sum_d \sum_j a_{edj} x_{dj}(w) \right), \quad \text{subject to (2.1) and (2.3).}$$

It is stated in [2] that the above problem is also NP-complete.

3. Direct approach

As shown in Section 2, the allocation task considered in this paper (FAP) is NP-complete. Hence, the heuristic methods are called for. In this section we present three heuristic algorithms for solving FAP based, respectively, on local search, simulated annealing and Lagrangean relaxation. We start, however, with the branch-and-bound approach.

3.1. Branch-and-bound

Let us first notice that FAP can be formulated as a mixed linear-integer programme, using the node–link formulation of the directed graphs multicommodity problems. The following formulation is due to Tomaszewski [6]. For the purpose of this section the usual notation used in this paper is slightly changed.

The given demand volume to be allocated from node v to node t is given by $d(v,t)$, and $o(e)$ and $t(e)$ denote the starting and end nodes of link e, respectively. Let V denote the set of nodes, let $w(e) \in [0,1]$ denote the weight of link e (variables), and let $W(v,t)$ be the length of the shortest path from v to t (variables). Let $\{\delta(e,t): e \in E, t \in V\}$ be a set of binary variables such that if $\delta(e,t) = 1$ iff link e is on a shortest path to node t. Let $f(e,t)$ (variables) denote the flow to node t on link e (it should be zero if e is not on a shortest path to node t). Let $f_x(v,t)$ (variables) denote the maximum flow to node t on all links outgoing from node v; this should be the flow on each link that is on a shortest path to node t.

FAP in the MIP formulation:

$$\sum_{e:o(e)=t} f(e,t) - \sum_{e:t(e)=t} f(e,t) = -\sum_{x \in V} d(x,t), \quad \forall t \in V, \tag{3.1}$$

$$\sum_{e:o(e)=v} f(e,t) - \sum_{e:t(e)=v} f(e,t) = d(v,t), \quad \forall t \in V,\ \forall v \in V, v \neq t, \tag{3.2}$$

$$\sum_{t \in V} f(e,t) \leq c(e), \quad \forall e \in E, \tag{3.3}$$

$$0 \leq f_x(o(e),t) - f(e,t) \leq (1 - \delta(e,t)) \sum_{v \in V} d(v,t), \quad \forall t \in V,\ \forall e \in E, \tag{3.4}$$

$$f(e,t) \leq \delta(e,t) \sum_{v \in V} d(v,t), \quad \forall t \in V,\ \forall e \in E, \tag{3.5}$$

$$0 \le W(t(e),t) + w(e) - W(o(e),t) \le (1-\delta(e,t))|V|, \quad \forall t \in V,\ \forall e \in E, \tag{3.6}$$

$$1 - \delta(e,t) \le (W(t(e),t) + w(e) - W(o(e),t))|V|, \quad \forall t \in V,\ \forall e \in E, \tag{3.7}$$

$$\sum_{e:o(e)=v} \delta(e,t) \ge 1, \quad \forall t \in V,\ \forall v \in V. \tag{3.8}$$

Note that using equality in (3.8) forces the shortest paths to be unique. Unfortunately, it turns out that the above MIP problem is difficult to solve already for small networks (we have tried CPLEX [7]) so we do not report any numerical results for (3.1)–(3.8).

3.2. Simulated annealing (SAN)

SAN is a well-known multi-purpose meta-heuristic for combinatorial optimisation (cf. [8]). For some problems SAN is able to find solutions close to global optima, even for the problems with large state spaces. The advantages of this heuristic are its general usability, easy adaptation to a particular application and easy implementation. For the network design purpose it has been applied, e.g. in [9]. In our implementation the integer weight systems (2.3) are assumed.

```
begin
  initialise(w_old); min_cost:= C(w_old); w_best:= w_old; T:= initial_temperature;
  while T ≥ temperature_lower_bound and min_cost > cost_lower_bound
    for counter:= 0 to counter_upper_bound do
      w_new:= neighbour(w_old); ΔC:= C(w_new)-C(w_old);
      if ΔC ≤ 0 then
      begin
        w_old:= w_new; if C(w_new) < min_cost then begin min_cost:= C(w_new);
          w_best:= w_new end
      end
      else if random < exp{−ΔC/T} then w_old:= w_new;
    end for;
    T:= T × a
  end while
end
```

We start with solution w_old generated by procedure initialise(w_old) and route all the demands accordingly. Then, at each step, the algorithm selects a neighbour of w_old, using function neighbour(w_old). The neighbouring state, w_new, is obtained by selecting at random a link and incrementing or decrementing its weight by 1 (the selection from the two possibilities is also random). Then the demands are routed according to the new weight system w_new, and the cost of the new state, C(w_new), is calculated and compared to the cost of the old one, C(w_old). If the cost of the new state is not greater than of the old one, the new state is always accepted. If C(w_new) is greater than C(w_old), the new state is accepted according to the *metropolis test*. The outcome of the test depends on the current temperature T (basic control parameter of SAN) and on the cost difference between the states (ΔC). At the beginning, SAN will accept states with relatively large ΔC with a high probability, which is then decreased exponentially during the optimisation process. This allows for a better scanning of the state space to avoid local optima.

At the end of the main loop the temperature is decreased ($a < 1$); the loop is executed until the temperature reaches a predefined lower bound. The cost of solution w is equal to the total exceeded capacity: $C(w) = \sum_e \max\{\sum_d \sum_j a_{edj} x_{dj}(w) - y_e, 0\}$.

3.3. Weights adjustment (WA)

The following local search method tries to directly compute a feasible link weight system. The method iteratively adjusts the links weights on the basis of the current links loads: in the consecutive steps the algorithm increases the weights of overloaded links and decreases the weights of the under-loaded ones. Two types of weight systems can be handled: integer systems (2.3) or continuous systems (2.4). The algorithm works according to the pseudo-code given below.

The algorithm starts from a randomly generated weight system w. If the network is overloaded the weight adjustment process is started. All demands are routed according to the current weight system w_old, and then the cost of each link, cost_new(e), is calculated as follows. If link e is under-loaded, its cost is made equal to its current load (occupied capacity) $\underline{y}_e$ (cost_new(e) = $\underline{y}_e$) and its weight is decreased by a small random value. If the link is overloaded, its cost becomes a sum of two terms: one equal to the capacity y_e, and a second equal to the square of the link load minus its capacity (cost_new(e) = $y_e + (\underline{y}_e - y_e)^2$); then the weight of the considered link is increased to remove it form some demand flows. The magnitude of the increase depends on the value of cost_new(e) and the absolute value of the difference between the cost_old(e) and cost_new(e).

If for the current weight system the network is under-loaded, the second part of the algorithm is activated. The most loaded link and the least loaded link are selected, and their weights are increased and decreased by 1, respectively. Then all demands are routed according to the adjusted weight system, and the network cost is calculated (the cost can reflect one of three possible objectives: maximisation of the average free capacity, maximisation of the total free capacity, or minimisation of the variance of the links free capacity). If the cost decreases (so the algorithm moves in the right direction) the new weight system is accepted, whilst if the cost increases, the system is accepted according to the Metropolis test of SAN (cf. Section 3.2). If the new weight system is accepted the procedure is repeated (provided all links are under-loaded).

```
begin
  for e:= 1 to E do set_counter(e); generate_initial_weight_system(w_old);
  for step:= 1 to max_step do
    begin
      route_demands(w_old);
      for e:= 1 to E do cost_old(e):= compute_cost(e);
      if network is overloaded then
      for e:= 1 to E do
        begin
          cost_new(e):= compute_cost(e);
          w_new(e):= modify_weight(cost_new(e), cost_old(e), w_old(e)); cost_old(e):= cost_new(e);
        end;
      route_demands(w_new);
      if network is under-loaded then
```

```
        repeat
          w_new:= modify_weights(w_old);
          route_demands(w_new);
        until new solution is not accepted or network is overloaded;
      w_old:= w_new;
    end {main for loop}
end
```

3.4. Lagrangean relaxation (LR)

Consider the following linear programming task, called OT, with no OSPF constraints on flows:

$$\text{maximise } C = \sum_e b_e \left(y_e - \sum_d \sum_j a_{edj} x_{dj} \right), \tag{3.9}$$

subject to

$$\sum_j x_{dj} = h_d, \quad d = 1, 2, \ldots, D, \tag{3.10}$$

$$\sum_d \sum_j a_{edj} x_{dj} \le y_e, \quad e = 1, 2, \ldots, E, \tag{3.11}$$

where $x_{dj} \ge 0$ is the flow realising demand d on path j, and b_e are given coefficients.

Using LR we can solve the problem dual to OT (cf. [10]). The idea of this approach is that the dual solution yields a weight system that can be used for the OSPF routing. The dual problem is obtained by dualising (4.3) and (4.4), and forming the Lagrangean:

$$\begin{aligned} L(\pi, \lambda, x) &= \sum_e b_e \left(y_e - \sum_d \sum_j a_{edj} x_{dj} \right) + \sum_d \lambda_d \left(h_d - \sum_j x_{dj} \right) + \sum_e \pi_e \left(\sum_d \sum_j a_{edj} x_{dj} - y_e \right) \\ &= \sum_d \lambda_d h_d - \sum_e (b_e + \pi_e) y_e + \sum_d \sum_j \left(\sum_e a_{edj} (b_e + \pi_e) - \lambda_d \right) x_{dj}. \end{aligned} \tag{3.12}$$

The dual problem to OT, abbreviated to DP, is as follows:

$$\text{maximise } W(\pi, \lambda) = \min_{x \ge 0} L(\pi, \lambda, x), \text{ over } \pi \ge 0, \text{ and } \lambda \text{ with unlimited sign.} \tag{3.13}$$

DP can be solved with subgradient optimisation (cf. [11]) since it can be shown that (3.13) is equivalent to

$$\text{maximise } V(\pi) = \sum_e (b_e + \pi_e)(\underline{y}_e - y_e) \text{ over } \pi \ge 0, \tag{3.14}$$

where $\underline{y}_e$ is the load of link e resulting from allocating each demand volume to one of its cheapest (shortest) path with respect to the link costs equal to $(b_e + \pi_e)$, $e = 1, 2, \ldots, E$. For a fixed π the subgradient is

calculated according to the formula:

$$\frac{\partial V(\pi)}{\partial \pi_e} = \underline{y}_e - y_e, \quad e = 1, 2, \ldots, E. \tag{3.15}$$

Alternatively, we can use the following formulation dual linear programming formulation DLLP:

$$\text{maximise } W(\pi, \lambda) = \sum_d \lambda_d h_d - \sum_e (b_e + \pi_e) y_e,$$

subject to

$$\lambda_d \leq \sum_e a_{edj}(b_e + \pi_e), \quad j = 1, 2, \ldots, m(d), \ d = 1, 2, \ldots, D, \tag{3.16}$$

$$\pi, \lambda \geq 0 \tag{3.17}$$

DLPP is more efficient in the cases when we can predefine the sets of allowable paths for the demands. Otherwise, for large networks, the subgradient solution of DP is usually superior to the LP solution since it can easily scan all the paths with the Dijkstra shortest path algorithm (to compute $\underline{y}_e$ for fixed π).

After solving LR we arrive at a set of optimal multipliers $\pi^0 = (\pi_1^0, \pi_2^0, \ldots, \pi_E^0)$ and define a weight system:

$$w_e^0 = b_e + \pi_e^0, \quad e = 1, 2, \ldots, E. \tag{3.18}$$

The system w^0 has a property that all the non-zero optimal primal flows (solving OT) can be realised only on the paths that are the shortest with respect to the weights w^0. Hence, if for each demand d there exist only one such a shortest path, then task FAP of Section 2 (and hence the OSPF routing problem) is solved. However, the uniqueness of the shortest paths is not guaranteed (cf. Section 6). If the uniqueness is not the case, we can anyhow try to use the weight system w^0 for the OSPF routing. This in general will lead to a non-feasible solution to FAP, since the flows that solve the primal problem AT4 are in general different than those generated with the ECMP rule. Nevertheless, if a number of demands with multiple shortest paths is not large and when the number of the shortest paths for such demands is low (2–3 shortest paths) then we can expect that the ECMP flows will give a good near-optimal solution.

4. Two-phase approach

In this section we formulate a two-phase approach, in which both phases are based on mathematical programming: Phase 1 on MIP, and Phase 2 on linear programming (LP). The idea is to allocate all demands to single paths (Phase 1) and then to try to find a weight system for which the paths realising demands are the unique shortest paths (Phase 2). An additional motivation behind the two-phase approach is that it leads to weight systems w with the property that for each demand there is only one, unique shortest path with respect to w. This allows for applying the simplest version of the Dijkstra shortest path algorithm at the nodes in order to set the packets' routing tables (otherwise more complicated algorithms have to be used). However, no effective necessary and sufficient condition, which can be used to generate such a subset of W, is available (cf. [3,4]). The approach, although simpler than the direct one, not always leads to a feasible solution, first of all because of possible non-feasibility of the resulting Phase 2 task.

4.1. Formulation of the two-phase optimisation task

1. Phase 1 (MIP)
   - indices:
     $d = 1, 2, \ldots, D$ demand
     $j = 0, 1, \ldots m(d)$ paths for flows realising demand d
     $e = 1, 2, \ldots E$ links
   - constants:
     h_d minimal volume of demand d
     a_{edj} 1 if e belongs to path j realising demand d, 0 otherwise
     y_e capacity of link e
   - variables:
     ε_{dj} binary variables forcing the single-path flow of demand d
   - constraints:

$$\sum_j \varepsilon_{dj} = 1, \quad d = 1, 2, \ldots, D, \tag{4.1}$$

$$\sum_d \left(\sum_j a_{edj} \varepsilon_{dj} \right) h_d \leq y_e, \quad e = 1, 2, \ldots, E. \tag{4.2}$$

Suppose we have found a solution to Phase 1. For the purpose of Phase 2 the paths are renumbered in order to make paths $j = 0$ the ones that carry the whole flow (i.e. to make $\varepsilon_{d0} = 1$ for each demand d).

1. Phase 2 (LP)
   - variables:
     w_e weight of link e (continuous variable)
   - constraints:

$$\sum_e a_{ed0} w_e \leq 1 + \sum_e a_{edj} w_e, \quad d = 1, 2, \ldots, D, \quad j = 1, 2, \ldots, m(d), \tag{4.3}$$

$$w_e \geq 1, \quad e = 1, 2, \ldots, E. \tag{4.4}$$

The above linear programme gives the necessary and sufficient conditions for existence of a weight system yielding the paths identified by $(d, 0)$ as the unique shortest paths. If the LP is infeasible then the weight system does not exist. Although, formally, the list $j = 0, 1, \ldots, m(d)$ should contain all the network paths for each demand d, the above LP can be solved without generating all constraints (4.3), as explained in [3].

4.2. Solving Phase 1

The variants of Phase 1 are formulated as MIPs and as such can be solved exactly only for small networks, as available exact methods for MIPs (i.e. the branch-and-bound and the cutting-plane methods) usually fail for large networks because of excessive time and memory requirements. Fortunately, in the considered single-path allocation case, approximate heuristic methods can be effective in terms of solutions quality (suboptimality) and of acceptable computation times.

Below we describe two such approaches for solving Phase 1: evolutionary algorithm and simulated allocation.

4.2.1. Evolutionary algorithm (EA)

The use of EAs is another well-known meta-heuristic [12]. In the context of network design it has been used in, e.g. [13,14]. Below we present pseudo-code of the so-called $(\mu + \lambda)$ evolution strategy that we use to solve Phase 1.

---

begin
  n:= 0; initialise(P_0);
  while not stop_criterion **do**
    $O_n = \varnothing$;
    for i:= 1 **to** λ **do** O_n:= $O_n \cup$ crossover(P_n); **for** $\boldsymbol{\varepsilon} \in O_n$ **do** mutate($\boldsymbol{\varepsilon}$);
    P_{n+1}:= select_best($O_n \cup P_n$);
    n:= $n + 1$
  end while
end

EA works with full allocation states $\boldsymbol{\varepsilon} = (\boldsymbol{\varepsilon}_{dj},\ d = 1, 2, \ldots, D,\ j = 0, 1, \ldots, m(d))$, called chromosomes, satisfying constraints (4.1). Within a chromosome, each subsequence $\boldsymbol{\varepsilon}_d = (\boldsymbol{\varepsilon}_{dj},\ j = 0, 1, \ldots, m(d))$ is called a gene (corresponding to demand d). Constraints (4.2) are taken into account via a penalty function, as will be shown below.

The algorithm starts with forming an initial population P_0 of μ chromosomes, each generated randomly, with all genes satisfying constraints (4.1). At each step n, a set O_n of λ chromosomes is formed. Each element of this set is obtained as an outcome of the crossover operations on two (parent) chromosomes of population P_n (parents). Each parent is selected from the population with the probability proportional to its fitness function

$$C(\boldsymbol{\varepsilon}) = \sum_e \left(\max \left\{ \sum_d \left(\sum_j a_{edj} \boldsymbol{\varepsilon}_{dj} \right) h_d - y_e, 0 \right\} \right)^2 . \tag{4.5}$$

Having fixed the parents, their off-spring is formed by taking gene by gene at random from the parents (gene $\boldsymbol{\varepsilon}_d$, $d = 1, 2, \ldots, d$, is taken from a parent with probability 1/2). Next, each chromosome from the so formed set O_n is mutated. The mutation consists in changing randomly the current allocation path in each gene of the mutated chromosome with a low probability p (e.g. $p = 1/D$).

Finally, the next population is formed by taking the best, according to the fitness function, elements out of the previous population P_n and the set O_n. The main step is repeated until the fitness function of the best chromosome in the current population is equal to 0, or there is no improvement in the consecutive N steps (i.e. the best chromosome has the same, positive value of the fitness function in N consecutive steps).

4.2.2. Simulated allocation (SAL)

SAL is a meta-heuristic which has been previously applied to similar problems [15]. It is particularly well suited for Phase 1 because of the assumed full demand aggregation (demand volumes are allocated to single paths). In fact, SAL is more effective than EA in finding feasible solutions of the Phase 1 problem.

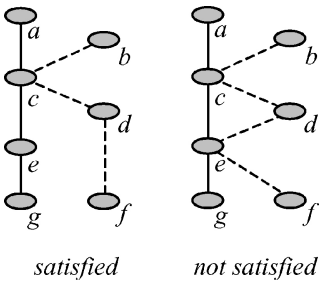


Fig. 4. A necessary condition.

Below we briefly describe a more sophisticated application of SAL to Phase 1. In the application any set of currently used paths fulfils a simple necessary uniqueness condition illustrated in Fig. 4. The condition (cf. [3,4]) requires that if two paths meet at a certain node, they must continue their way along a common sequence of links until they split for good (in Fig. 4 the paths *a–c–e–g* and *b–c–d–f* on the left side satisfy this condition, whilst the paths *a–c–e–g* and *b–c–d–e–f* to the right do not). The condition is known to be rather powerful [3,4]. Note that for undirected graphs it is easy to check the condition for a given pair of paths: if the paths are not disjoint, the number of common nodes must be equal exactly to the number of common links plus 1.

The SAL algorithm works with partial allocation flow sequences (states) $\boldsymbol{\varepsilon} = (\boldsymbol{\varepsilon}_{dj}, d = 1, 2, \ldots, D, j = 1, 2, \ldots, m(d))$, i.e. (4.1) does not necessarily hold for all demands. The algorithm starts with the all zero-flow solution ($\boldsymbol{\varepsilon}_{dj} \equiv 0$) and in each step chooses, with probability $q(\boldsymbol{\varepsilon})$, between allocate ($\boldsymbol{\varepsilon}$), i.e. adding one demand flow to the current flow sequence $\boldsymbol{\varepsilon}$, and *disconnect* ($\boldsymbol{\varepsilon}$), i.e. removing one or more demand flows from the current solution $\boldsymbol{\varepsilon}$. We require that $q(\boldsymbol{\varepsilon}) > 1/2$, except for maximal allocation states $\boldsymbol{\varepsilon}$ for which $q(\boldsymbol{\varepsilon}) = 0$ ($\boldsymbol{\varepsilon}$ is a maximal allocation state if all demands are allocated, i.e. when $|\boldsymbol{\varepsilon}| = \sum_d \sum_j \boldsymbol{\varepsilon}_{dj} = D$). Whenever a complete allocation state is reached, a check is made whether the cost of the best reached so far solution (min_cost) is improved. Procedure *disconnect*($\boldsymbol{\varepsilon}$) is used in two variants:

disconnect_1($\boldsymbol{\varepsilon}$): remove from $\boldsymbol{\varepsilon}$ one previously allocated demand flow (at random);
disconnect_2($\boldsymbol{\varepsilon}$): remove from $\boldsymbol{\varepsilon}$ all the demand flows which use all the overloaded links, and, additionally, some randomly chosen links (with a certain probability).

---

begin
  step:= 0; min_cost:= ∞; $\boldsymbol{\varepsilon}$:= 0;
  repeat
    step:= step + 1;
    if *random* < $q(\boldsymbol{\varepsilon})$ **then** *allocate*($\boldsymbol{\varepsilon}$) **else if** $C(\boldsymbol{\varepsilon})$ < min_cost **then** disconnect_1($\boldsymbol{\varepsilon}$)
      else disconnect_2($\boldsymbol{\varepsilon}$);
    if $\boldsymbol{\varepsilon}$ is a maximal allocation state **and** $C(\boldsymbol{\varepsilon})$ < min_cost **then**
      begin min_cost:= $C(\boldsymbol{\varepsilon})$; $\boldsymbol{\varepsilon}$_best:= $\boldsymbol{\varepsilon}$ **end**
  until step = step_limit **or** min_cost = cost_lower_bound
end

The second variant is applied if (and only if) the maximal allocation state is reached or if the current auxiliary cost $C(\boldsymbol{\varepsilon}) = \sum_e \max\{\sum_d (\sum_j a_{edj}\varepsilon_{dj})h_d - y_e), 0\}$ (expressing the total exceeded links capacity in the current allocation state) becomes greater than the current value of min_cost.

Procedure *allocate*($\boldsymbol{\varepsilon}$) assigns demand flow h_d to a selected path and increments the corresponding entry ε_{dj} by 1. The demand d to be allocated is chosen at random from the set of the not yet allocated demands; for a given d, the allocation path j is selected using a shortest path algorithm (see below). It is important that a new demand can be allocated only to a path that satisfies, together with the allocation paths used in the current solution $\boldsymbol{\varepsilon}$, the necessary feasibility condition described at the beginning of this section.

In fact, the allocation probability $q(\boldsymbol{\varepsilon})$ depends on the state through the number of allocated demands, i.e. $q(\boldsymbol{\varepsilon}) = q(|\boldsymbol{\varepsilon}|)$. Of course, $q(0) = 1$ and $q(D) = 0$. One way to settle the allocation probabilities is to choose a threshold $\underline{D}(0 \leq \underline{D} \leq \underline{D}$, e.g. $\underline{D} = 0.8D$), and to set $q(k) = 1$ for $k \leq \underline{D}$ and $q(k) = \underline{q}$ for $\underline{D} < k < D$, for some fixed $\underline{q} > 1/2$.

The algorithm is terminated either when a feasible solution to Phase 1 allocation problem is found or when the assumed limit on the number of steps (executions of the main loop) is reached.

To find an allocation path for the currently selected demand, any standard shortest path labelling algorithm can be used (e.g. the Dijkstra algorithm) with a modified way of nodes labelling. The modification affects the way new nodes are labelled from the already labelled ones. Consider a graph with undirected links. For each node pair $\{a, b\}$ there is specified an attribute $n(a, b)$ equal to the number of times the two nodes belong to the same path in the set of the currently allocated paths. When a demand is removed, the attribute $n(a, b)$ is decremented by for all node pairs $\{a, b\}$ belonging to the path being just disconnected.

When a demand between nodes s and t is to be allocated, the shortest path tree starting from node s is built according to the standard labelling rule with the following, important adjustments. Suppose we are at a labelled node a, and node a is on the path s–c–$\cdots$–d–f–$\cdots$–a from nodes in the tree under construction, and we consider labelling node b from node a. Then we can label node b only when (cf. Fig. 5) the following cases hold.

Case 1. Nodes a and b do not belong to a common path ($n(a, b) = 0$): The path s–c–$\cdots$–d–f–$\cdots$–a in the currently constructed tree cannot contain any node v belonging to a common allocation path with node $b(n(v, b) = 0$ for $v = s, c, \ldots, d, f, \ldots, a)$.

Case 2. Nodes a and b belong to at least one common allocation path ($n(a, b) > 0$): The path s–c–$\cdots$–d–f–$\cdots$–a in the currently constructed tree has the following property: nodes from s to d do

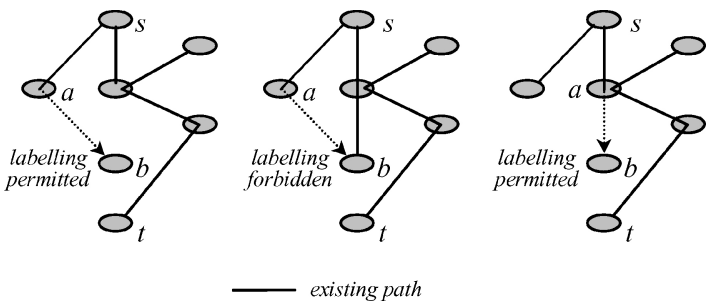


Fig. 5. Node labeling.

not belong to a common allocation path with node b, and f–$\cdots$–a–b is a subpath of one of the currently allocated paths. The latter condition is equivalent to: edge $\{a, b\}$ belongs to at least one path and nodes f, a and b belong to at least one common path.

If node t is reached, an allocation path is found and the demand volume between nodes s and t is allocated to it. Then the attribute $n(a, b)$ is incremented by 1 for all node pairs $\{a, b\}$ belonging to the selected path.

When demand d is being allocated in state $\boldsymbol{\varepsilon}$, the metric of link e used in the labelling decisions is equal to $1 + T(\max\{0, \underline{y}_e(\boldsymbol{\varepsilon}) + h_d - y_e\})^2 + b_e$. In the preceding definition $\underline{y}_e(\boldsymbol{\varepsilon})$ is the load of link e in state $\boldsymbol{\varepsilon}$ (i.e. $\underline{y}_e(\boldsymbol{\varepsilon}) = \sum_d(\sum_j a_{edj}\varepsilon_{dj})h_d)$), T is a fixed positive number. The definition of b_e, which takes the history of the process into account, is somewhat more complicated: b_e is incremented each time the capacity of link e is exceeded during optimisation. Still, b_e is bounded—its value cannot exceed the upper bound B (if b_e reaches bound B then it stays with this value).

The above described labelling procedure works as well for directed graphs. Note, however, that in both cases there may appear situations when a new path cannot be allocated because the existing consistent set of paths blocks all the paths from s to t. An example presented in Fig. 6 [6] depicts such a situation: all paths from s to t are inconsistent with the consistent set of paths $\{p_1, p_2, \ldots, p_6\}$. Such situations are the more frequent the higher the ratio D/E. Fortunately, due to the stochastic character of SAL, when a blocking situation is encountered for some current set P of allocated paths and for some demand d, sooner or later demand d will be allocated because the current set P is changing all the time in a stochastic manner.

4.3. Solving Phase 2

The linear programme (4.3)–(4.4) for Phase 2 can be effectively solved by considering only the meaningful constraints through consecutive generating of two shortest paths for each demand [3]. An alternative formulation is discussed in [4]. Below we give still another necessary and sufficient condition derived from the dual theory [10], especially useful for testing a system of paths for not being realisable.

There exists a system of weights $w = (w_1, w_2, \ldots, w_E)$ with $w_e \geq 0$ and $\sum_e w_e = 1$ such that for each demand $d = 1, 2, \ldots, D$ the path $(d, 0)$ is the unique shortest path if and only if the following LP

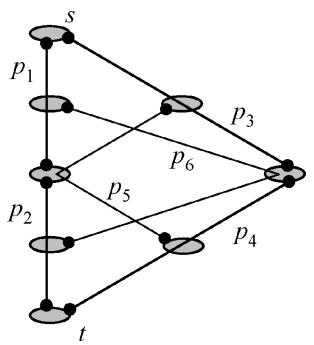


Fig. 6. A blocking situation in SAL.

- constants:

$$r_{edj} = \begin{cases} 0 & \text{if both path number 0 and path number } j\ (j = 1, 2, \ldots, m(d)) \text{ realising demand } d \text{ contain link } e, \text{ or both paths do not contain link } e \\ 1 & \text{if path } j\ (j = 1, 2, \ldots, m(d)) \text{ realising demand } d \text{ contains link } e \text{ and path number 0 does not} \\ -1 & \text{if path } j\ (j = 1, 2, \ldots, m(d)) \text{ realising demand } d \text{ does not contain link } e \text{ and path number 0 does} \end{cases}$$

- variables:
  σ_{dj} non-negative continuous variable
- constraints:

$$\sum_{d}\sum_{j>0} r_{edj}\sigma_{dj} \le 0, \quad e = 1, 2, \ldots, E, \tag{4.7}$$

$$\sum_{d}\sum_{j>0} \sigma_{dj} = 1 \tag{4.8}$$

is infeasible.

5. Numerical results

Below we discuss applications of the algorithms described in Sections 3 and 4 for two sets of network configurations: a set of two undirected 7-node (N7) and a 12-node (N12) networks and a set of four 7, 14, 28 and 56-node directed networks. We start with the undirected networks. N7 is an artificial network consisting of seven nodes, $E = 12$ links and $D = 21$ demands, whilst N12 is a model of a Polish transit long-distance network and consists of 12 nodes, $E = 18$ links and $D = 66$ demands (cf. Fig. 7). Individual demand volumes for N12 and N7 are given in Table 1.

As mentioned above, in N7 and N12 demands, links and paths are bidirectional. For N7 we consider one set of links capacities, and for N12—two such sets (coded by links end-nodes). The considered configurations are saturated in the sense that the demand realisation consumes all available links capacity.

N7-1 : $y(1,2) = 75$, $y(1,6) = 75$, $y(1,7) = 79$, $y(2,3) = 78$, $y(2,7) = 87$, $y(3,4) = 77$, $y(3,7) = 97$, $y(4,5) = 71$, $y(4,7) = 100$, $y(5,6) = 80$, $y(5,7) = 95$, $y(6,7) = 86$,

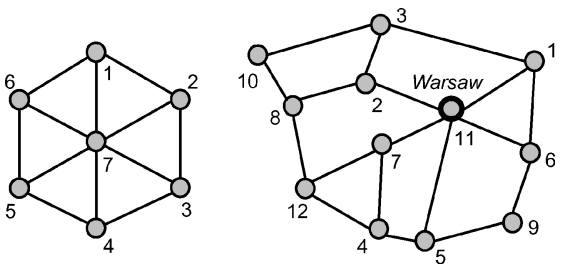


Fig. 7. Graphs of N7 and N12.

Table 1
Demand volumes

	N12											N7					
	2	3	4	5	6	7	8	9	10	11	12	2	3	4	5	6	7
1	10	11	13	13	16	7	19	10	6	13	14	24	25	28	19	49	32
2	0	3	21	19	21	14	32	14	5	68	32	0	36	17	38	26	49
3	0	0	27	21	25	17	35	14	3	107	24	0	0	31	46	46	34
4	0	0	0	11	21	15	51	21	19	84	40	0	0	0	24	25	31
5	0	0	0	0	13	20	35	5	18	74	28	0	0	0	0	36	33
6	0	0	0	0	0	16	34	4	9	30	18	0	0	0	0	0	21
7	0	0	0	0	0	0	21	14	12	4	15						
8	0	0	0	0	0	0	0	28	47	129	14						
9	0	0	0	0	0	0	0	0	7	61	13						
10	0	0	0	0	0	0	0	0	0	24	19						
11	0	0	0	0	0	0	0	0	0	0	97						

N12-1 : $y(1,11)=89$, $y(1,3)=17$, $y(2,3)=273$, $y(2,8)=391$, $y(2,11)=545$, $y(3,10)=9$, $y(4,5)=216$, $y(4,7)=146$, $y(4,12)=287$, $y(5,9)=122$, $y(5,11)=127$, $y(6,9)=137$, $y(6,11)=242$, $y(7,11)=284$, $y(7,12)=177$, $y(8,10)=160$, $y(8,12)=328$, $y(1,6)=26$,

N12-2 : $y(1,11)=47$, $y(1,3)=135$, $y(2,3)=205$, $y(2,8)=32$, $y(2,11)=376$, $y(3,10)=211$, $y(4,5)=275$, $y(4,7)=15$, $y(4,12)=241$, $y(5,9)=67$, $y(5,11)=317$, $y(6,9)=124$, $y(6,11)=221$, $y(7,11)=382$, $y(7,12)=338$, $y(8,10)=236$, $y(8,12)=379$, $y(1,6)=128$.

Table 2 summarises the performance of the methods for the above network examples. For all methods, except for LR which is not a stochastic method, the time (for the following computers: WA and SAN—Sparc I 143 MHz; LR and EA—Sparc II 250 MHz; SAL—PC 366 MHz) and the number of steps presented

Table 2
Performance of the methods

	Link (1)		Load (2)		Time (s) (3)			Steps (4)		
	All	LR	All	LR	WA/C	WA/I	SAN	WA/C	WA/I	SAN
N7-1	2	–	23	–	0.18	0.06	<30	17	7	<4000
N12-1	0	0	0	0	0.24	0.15	<30	26	11	<5000
N12-2	1	2	13	34	1.12	0.67	<30	49	36	<5000
	EA	SAL	EA	SAL	LR	EA	SAL	LR	EA	SAL
N7-1	0*	2	0*	23	0.5	4	0.1	<7000	5000	130
N12-1	0	0	0	0	0.03	129	9.3	<20	30000	6000
N12-2	0*	1	0*	13	4.4	174	6.8	<7000	40500	15000

in columns 3 and 4 are averages over several tens of runs for each case, and they give the values for entering the reported solutions for the first time during a single run. The notion of "step" has different interpretation for different methods. In WA one step consists of computing all flows for a new weight system from scratch, while for SAN and SAL one step requires less computations, as only few flows have to be recomputed. For EA one step means the computation of links loads for one chromosome. For LR one step corresponds to one computation of the subgradient of the dual function.

The number of overloaded links is shown in column 1 whilst the total links load exceeding the total links capacity is shown in column 2. The three direct methods WA/C (continuous weights), WA/I (integral weights) and SAN, and the two-phase method based on SAL found the same final solutions in terms of the demand flows. For each network all demands were allocated to single paths (for SAL this is an intrinsic feature); the weight systems, however, were different for different methods (recall that for SAL the weights are found in Phase 2 by solving an LP task). For N12-1 the direct method LR quickly found a weight system that implies unique shortest paths (identical to those found by all other methods from Table 2) and thus solved the OSPF allocation problem. For N12-2, LR found systems of weights that imply unique shortest paths for 61 demands, still for the remaining five demands exactly two shortest paths appeared. Applying CPLEX we found the corresponding flows (demand split) for the demands with more than one shortest paths, finding a solution to FAP. The split is different from the ECMP split, and therefore the LR solution does not strictly solve the original problem. However, if we use the weights of LR and apply the ECMP rule, we arrive at the flows which overload only two links and yield a solution not much worse than that of WA, SAN and SAL. Unfortunately, this scheme does not work well for N7. Here, only 10 out of 21 demands have unique shortest paths, and there is a demand with as many as seven shortest paths. In consequence, if we calculate the ECMP flows for the LR weights then many links will be overloaded (and many under-loaded) so the LR approach simply does not work (this is indicated with "–" in columns 1 and 2). Despite these limitations, we note that LR, as a scalable method applicable for large networks, may be used to yield initial weight systems (starting points) for other direct methods (as WA and SAN).

We have also applied the two-phase approach with EA in Phase 1. In all the three cases the resulting single path allocations were feasible in terms of Phase 1 (all links were saturated and no link was overloaded). For N12-1 the solution was identical with the one found by all other considered methods. For N7-1 and N12-2, however, the EA solutions could not be realised with any weight system (they are not feasible OSPF solutions—this is marked with "*" in columns 1 and 2). Nevertheless, as EA is a powerful (although time consuming) method for finding feasible single path demand allocations (independent of the ECMP constraints), it can be useful for finding lower bounds for link overloads in the cases when a feasible ECMP solution cannot be found by other methods. Also if EA cannot find any feasible solution in Phase 1, then also the original problem (FAP) will most likely be non-feasible.

Next we have considered a set of four directed (artificial) networks with 7, 14, 28 and 56 nodes (cf. Table 3). Each network was dimensioned using a design version of WA in three variants: saturated (realisation of demands consumes all available links capacity), and with medium and with high over-dimensioning of the links. The results are reported for WA/I, SAN and SAL, and, additionally for uniform weights (UWs) and inverse capacity weights (ICWs).

Table 4 summarises the performance of the methods for the saturated examples. For all the methods, except for UW and ICW, which are not stochastic, the reported values are averages over several runs. The notation is as follows: AVRT—the average running time (s) for reaching the reported solutions for the first time during a single run, POPT—the percent of feasible solutions found in all runs, AVOL—the

Table 3
Example networks

	No. of links	Total capacity	Total capacity (medium over-dimension)	Total capacity (high over-dimension)	No. of demands	Total demand	Nodes/links
7 nodes	22	48000	52164	55968	42	34320	0.32
14 nodes	42	329280	342342	352308	182	172320	0.33
28 nodes	84	2192744	2283988	2347868	756	892492	0.33
56 nodes	224	9784716	9863190	11040272	3080	3340320	0.25

average number of overloaded links, and AVTO—the average total network overload (the sum of the exceeded links capacities). For UW and ICW the execution times are negligible.

Results for the remaining two cases are given in Tables 5 and 6. As the tables show, for all the considered cases the simple methods UW and ICW are strongly outperformed by the stochastic algorithms. For small networks WA, SAN and SAL are comparable in terms of the quality of results and the computation time (only SAN is more time consuming) for all the considered cases.

For large and tightly dimensioned networks the best results have been obtained with SAN (at the expense of long computation times). The algorithm, when appropriately tuned, gives almost deterministic results. This is true for example for the 56-node networks with the running time set to 1/2 h. With longer execution time (of about $2\frac{1}{2}$ h) the algorithm is also almost deterministic and sometimes it finds better results in terms of the unused network capacity in the cases of over-dimensioned networks. WA also gives results of good quality for the saturated and over-dimensioned networks; it is very fast and performs slightly better than SAL. For the saturated 28-node network, SAL was not able to find any feasible solution; WA found feasible solutions in almost 50% of runs. Note, however, that for the 56-node saturated network no algorithm was able to find any feasible solution. For more realistic cases of slightly over-dimensioned networks, WA and SAL produce results comparable with SAN, even for the 28-node network, but for 56 nodes the SAN is still the best (but, again, at the expense of very long computation times).

In terms of the computation times, WA is clearly superior to all the other stochastic methods. This method also gives good, consistent results for large networks and is very simple. SAL cannot currently compete with WA in terms of the execution times and the capability of computing near-optimal solutions. Note, however, that in case of the high over-dimensioning networks SAL gives results comparable to the other two methods, even for the 56-node network; SAL was able to find a feasible solution in 30% of runs, comparing to 10% for WA (this is also the case for the highly over-dimensioning cases). The slow performance of SAL for large networks is caused by the phenomenon of blocking states, for which the algorithm is not able to find a new path consistent, in the sense of the necessary condition for the existence of a weight system, with the already allocated paths. In large networks with long paths this phenomenon can be observed more often, making the method less effective (an improvement here can be achieved by finding a more effective way of omitting the blocking states—the work is in progress). We have also tested all the methods for the same network configurations, but dimensioned with a variant of SAL. The algorithm finds the first realisation of demands consistent in the sense of the necessary condition, and then computes the corresponding link capacities. The resulting networks are characterised in Table 7 (the difference in the number of links with the previous examples comes from the fact, that links with zero capacity are "erased"). For these networks the over-dimensioning was assured simply by adding 5 and 10% of capacity to each link. Note that, except for the 56-node network, the total capacity is greater

Table 4
Results for the saturated networks

	7 node network					14 node network					28 node network					56 node network				
	WA	SAN	SAL	UW	ICW	WA	SAN	SAL	UW	ICW	WA	SAN	SAL	UW	ICW	WA	SAN	SAL	UW	ICW
AVRT	0.03	–	0.06	0	0	0.2	–	1.31	0	0	13.7	–	198.96	0	0	155.8	–	1352	0	0
POPT	100	100	100	0	0	100	100	100	0	0	48	100	0	0	0	0	0	0	0	0
AVOL	0	0	0	9	12	0	0	0	20	15	4.3	0	7.4	45	45	61	18	85	127	86
AVTO	0	0	0	5280	10320	0	0	0	68520	85680	720	0	9096	438292	40338	1218	16912	24801	24890	33022

Table 5
Results for medium over-dimensioned networks

	7 node network					14 node network					28 node network					56 node network				
	WA	SAN	SAL	UW	ICW	WA	SAN	SAL	UW	ICW	WA	SAN	SAL	UW	ICW	WA	SAN	SAL	UW	ICW
AVRT	0.76	–	0.09	0	0	3.31	–	1.43	0	0	19.26	–	47.38	0	0	155.77	–	3211.2	0	0
POPT	100	100	100	0	0	100	100	100	0	0	100	100	100	0	0	0	0	0	0	0
AVOL	0	0	0	11	10	0	0	0	17	20	0	0	0	45	40	12	4	33.3	118	80
AVTO	0	0	0	9996	10236	0	0	0	38160	57144	0	0	0	356898	340480	26658	2028	114411	2024430	2637990

Table 6
Results for highly over–dimensioned networks

	7 node network					14 node network					28 node network					56 node network				
	WA	SAN	SAL	UW	ICW	WA	SAN	SAL	UW	ICW	WA	SAN	SAL	UW	ICW	WA	SAN	SAL	UW	ICW
AVRT	0.79	–	0.06	0	0	3.19	–	1.22	0	0	201.21	–	16.95	0	0	156.54	–	2916.5	0	0
POPT	100	100	100	0	0	100	100	100	0	0	100	100	100	0	0	10	100	30	0	0
AVOL	0	0	0	11	7	0	0	0	15	17	0	0	0	37	29	3	0	9.9	114	75
AVTO	0	0	0	10644	12456	0	0	0	25968	32556	0	0	0	290211	277721	2476.7	0	27458.4	2376990	2803860

Table 7
Parameters of the re-dimensioned networks

	No. of links	Total capacity	No. of demands	Total demand	Nodes/links
7 nodes	15	61080	42	34320	0.47
14 nodes	36	367680	182	172320	0.39
28 nodes	80	2260184	756	892492	0.35
56 nodes	208	9776880	3080	3340320	0.27

Table 8
Results for 56-node re-dimensioned network

	Saturated			Capacity + 5%			Capacity + 10%		
	WA	SAN	SAL	WA	SAN	SAL	WA	SAN	SAL
AVRT	149.81	–	1604.54	152.54	–	1320.61	153.73	–	1090.19
POPT	0	0	0	100	100	0	100	100	90
AVOL	61	8	72.5	0	0	4	0	0	1.1
AVTO	488.1	2520	114912	0	0	8608.2	0	0	0.18

than in the previous examples, although the number of links is always lower than for the networks of Table 3.

For these cases all stochastic algorithms give comparable results for 7, 14 and 28 node networks, either saturated or over-dimensioned. The difference in quality of the results can only be seen in the case of 56-node network. The results for this case are summarised in Table 8.

Finally, let us note that the paths yielded by SAL are almost always (except for one case) realisable with a weight system.

6. Conclusions

In the paper we have formulated an OSPF-related flow allocation problem (FAP) and proposed a set of methods for solving it. We have shown that FAP is NP-complete. We have considered a formulation of FAP in an MIP form; unfortunately the formulation turns out to be very difficult to solve already for small networks, even for such a sophisticated solver as CPLEX. Therefore, heuristic methods have to be applied for FAP.

Numerical studies indicate that the one-phase approach, called WA, consisting in direct finding of feasible OSPF link weight systems using a Local Search method, is quite effective and fast, still it requires some tunning. The SAN application finds in general better solutions than WA, still its running times are much longer.

We have also proposed a two-phase approach. In Phase 1 demands are allocated, using the SAL heuristic, to single paths so that the resulting set of paths fulfils a necessary condition for the existence of a weight system, according to which these paths are the unique shortest paths in the network graph. The desired weight system is then found in Phase 2 using LP. In effect, the two-phase approach is able to provide non-split flow solutions, which is an attractive feature for the network operation. For the series of the

directed networks considered in Section 5, the two-phase approach works nicely for networks up to 28 nodes. For the 56-node network SAL becomes less effective. The reason is that the number of demands increases with the square of the number of nodes, but the number of links increases linearly with this number. This makes difficult for SAL to consistently allocate new demands in the high allocation states (there are relatively too little paths per demand, and, at the same time, they are longer). However, in the real IP networks the number of transit nodes is not high (30 transit nodes, say) and the rest of the nodes are just ingress/egress routers which do not transit traffic. Thus, the two-phase approach based on the SAL heuristic could perhaps be quite useful in practice.

Finally, it should be noted that the proposed methods yield the weight systems that outperform the simple weight systems currently used in the OSPF networks (UWs or ICWs) to a considerable extent.

References

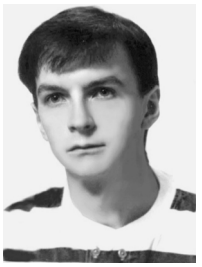
[1] OSPF Version 2, RFC2328 (http://www.ietf.org/rfc/rfc2328.txt), 1998.
[2] B. Fortz, M. Thorup, Internet traffic engineering by optimizing OSPF weights, in: Proceedings of the INFOCOM'2000, Tel Aviv, 2000.
[3] W. Ben Ameur, E. Gourdin, B. Liau, Dimensioning of internet networks, in: Proceedings of the DRCN'2000, Munich, 2000.
[4] A. Faragó, B. Szviatovszki, Á. Szentesi, Allocation of administrative weights in PNNI, in: Proceedings of the NETWORKS'98, Sorrento, 1998.
[5] M.R. Garey, D.S. Johnson, Computers and Intractability—A Guide to the Theory of NP-Completeness, Freeman, New York, 1979.
[6] A. Tomaszewski, August 2000, Private communication.
[7] ILOG CPLEX 6.5 Reference Manual, ILOG S.A., 1999.
[8] D.S. Johnson, C.R. Aragon, L.A. McGeoch, C. Schevon, Optimization by simulated annealing: an experimental evaluation, Oper. Res. 39 (1) (May–June 1991).
[9] M. Pióro, T. Stidsen, A. Glenstrup, C. Fenger, H. Christiansen, Design of robust optical networks, in: Proceedings of the NETWORKS'2000, Toronto, 2000.
[10] L.S. Lasdon, Optimization Theory for Large Systems, Macmillan, New York, 1970.
[11] M. Held, P. Wolfe, H. Crowder, Validation of subgradient optimization, Math. Program. 6 (1974).
[12] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer, Berlin, 1966.
[13] S. Kozdrowski, M. Pióro, J. Arabas, M. Szcześniak, Robust design of multicommodity integral flow networks, in: Proceedings of the Seventh International Conference on Genetic Algorithms ICGA'97, University of Michigan, East Lansing, 1997.
[14] D. Medhi, D. Tipper, Some approaches to solving a multi-hour broadband network capacity design problem with single-path routing, Telecommun. Systems 13 (2000) 269–271.
[15] M. Pióro, P. Gajowniczek, Solving multicommodity integral flow problems by simulated allocation, Telecommun. Systems 7 (1–3) (1997) 17–28.

M. Pióro is a professor at the Institute of Telecommunications, Warsaw University of Technology and at the Department of Communication Systems, Lund University. He received a Ph.D. degree in telecommunications in 1979 and a D.T.Sc. Degree in 1990, both from the Warsaw University of Technology. During 1984–1987 and 1997 he has been with the Lund Institute of Technology leading research projects for Ericsson Telecom AB in network design. In 1986 and 2001 he served as a senior expert for ITU. During 1990–1991 he had been working for Alcatel Standard Electrica, Spain, as a consultant in dynamic routing strategies. Currently he cooperates with Ericsson Traffic Laboratory in Budapest on IP network design problems. He has written two monographs and almost 100 papers presented in international telecommunications journals and conference proceedings. He is a technical editor of IEEE Communications Magazine. His research interests concentrate on modelling, performance evaluation and design of telecommunications networks.

Á. Szentesi received his M.Sc. degree in 1994 at the Budapest University of Technology and Economics, where he is currently finishing his Ph.D. Meanwhile, he also works as a research fellow at Ericsson Traffic Analysis and Network Performance Laboratory in Budapest, Hungary. His main interests are network planning and performance optimisation problems in high speed networks.

J. Harmatos received his M.Sc. degree in Electrical Engineering in 1998 at Budapest University of Technology and Economics. Currently he is a Ph.D. student at Department of Telecommunication and Telematics. He also works as a researcher at Ericsson Traffic Analysis and Network Performance Laboratory in Budapest, Hungary. His research areas are planning and optimisation of IP/OSPF. He is also interested in optimisation of the topology of UMTS networks.

A. Jüttner received his M.Sc. degree in mathematics in 1998 at the Eötvös Loránd University of Budapest, where he is currently working for his Ph.D. at Operational Research Department. He also works as a research fellow at Ericsson Traffic Analysis and Network Performance Laboratory in Budapest, Hungary. His main interests are combinatorial optimisation and its applications.



Piotr Gajowniczek was born in 1968 in Warsaw. He received his M.Sc. degree in control engineering in 1993 from the Warsaw University of Technology. Since then he works as a research assistant at the Institute of Telecommunications, Warsaw University of Technology. His interests concentrate around telecommunications network design and optimisation.

Stanislaw Kozdrowski was born in Ustrzyki Dolne, Poland, 1969. He received the M.Sc. degree in electronic engineering and Ph.D. degree in telecommunication from Warsaw University of Technology, Poland, in 1994 and 2000, respectively. From October 1998 to May 1999, he was a Fellow in the Institute of Telecommunication Systems of Lund University, Sweden. His research interests include optimisation and robust design of telecommunication networks by heuristic methods. He currently works for PTC (ERA GSM, Polish mobile operator) as an expert in telecommunication network dimensioning.