On OSPF Related Network Optimisation Problems

M.Pióro\textsuperscript{a,}\textsuperscript{b,}\textsuperscript{c}, Á.Szentesi\textsuperscript{a}, J.Harmatos\textsuperscript{a}, A.Jüttner\textsuperscript{a}, P.Gajowniczek\textsuperscript{c}, S.Kozdrowski\textsuperscript{c}

Ericsson Traffic Laboratory, Budapest, Hungary\textsuperscript{a}
Department of Communication Systems, Lund Institute of Technology, Sweden\textsuperscript{b}
Institute of Telecommunications, Warsaw University of Technology, Poland\textsuperscript{c}

address: Prof. Michal Pióro, Institute of Telecommunications, Warsaw University of Technology,
Nowowiejska 15/19, 00-665 Warszawa, Poland
tel: +48 22 825 98 20, fax: +48 22 660 7564, e-mail: mpp@tele.pw.edu.pl

Abstract
The paper deals with flow allocation problems in IP networks using OSPF routing. The main considered issue is the settlement of an OSPF link weight system in order to achieve near-optimal network throughput for the assumed (estimated) demand pattern and/or the availability of network resources. This 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/or with major resources’ failures.

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

1. Introduction
The OSPF (Open Shortest Path First) 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 more than one 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 the OSPF link-state flooding mechanism, all the nodes are aware of the weights \( w=(w_1,w_2,...,w_E) \) 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 is split according to the ECMP rule, not allowing for any flexibility during normal network operation. On the other hand, the weight system \( w \) can be adjusted and made fit to the actual network state and traffic conditions. 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 [2,3,4].
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 simple proof showing that the OSPF flow allocation problems considered in the paper are \( \text{NP} \)-complete [5]. The proof is given for the pure OSPF flow allocation problem, still it is valid for its extensions taking into account an objective function. These extensions are formulated in Section 3. Because of the \( \text{NP} \)-completeness of the OSPF flow allocation problems, one is forced to use heuristic methods for solving the problems. In Section 4 we discuss heuristic algorithms based on the search in the weight systems space using Local Search, Simulated Annealing and Lagrangean Relaxation. Next, in Section 5, 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, 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 6) and draw some conclusions (Section 7).

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. In other words, 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 consider the following OSPF Flow Allocation Problem (FAP).

FAP:

\[\text{indices:}\]
\[d=1,2,\ldots,D \quad \text{demands}\]
\[j=0,1,\ldots,m(d) \quad \text{paths for flows realising demand } d\]
\[e=1,2,\ldots,E \quad \text{links}\]

\[\text{constants:}\]
\[h_d \quad \text{volume of demand } d \text{ to be realised}\]
\[a_{edj} = 1 \text{ if link } e \text{ belongs to path } j \text{ realising demand } d, 0 \text{ otherwise}\]
\[y_e \quad \text{capacity of link } e\]

\[\text{variables:}\]
\[w_e \quad \text{weight of link } e \text{ (positive continuous variable)}, \ w = (w_1,w_2,\ldots,w_E)\]

\[\text{constraints:}\]
\[\sum_j x_{dj}(w) = h_d \quad d=1,2,\ldots,D \quad (2.1)\]
\[\sum_d \sum_j a_{edj} x_{dj}(w) \leq y_e \quad e=1,2,\ldots,E \quad (2.2)\]
\[w \in W. \quad (2.3)\]
Above, $x_d(j)(w)$ denotes the flow realising demand $d$ on path $j$, implied by the links’ weight system $w$. For a given weight system $w$, the flows $x_d(j)(w)$ are computed according to the ECMP rule. The rule is illustrated in Figure 1 above: the shortest paths $s-a-c-t$ and $s-a-d-t$ realise 0.25 of the total demand volume between nodes $s$ and $t$, whilst the shortest path $s-b-e-t$ realises the remaining 0.5 of the volume. Note that the flow functions $x_d(j)(w)$ are not given explicitly. In order to make the ECMP flow splitting procedure consistent, OSPF assumes that the link weights are positive, so there can be no loops in the shortest paths. Consequently, constraints (2.1) in FAP guarantee that all of the demands are realised, and constraints (2.2) - that links’ loads do not exceed their capacities. The weight systems’ space $W$ in constraint (2.3) is used to limit the set of values that can be assumed by the link weights systems; for instance, a state space assuring the consistency of the weight systems is:

$$1 \leq w_e \leq K \text{ and } w_e - \text{integer}, \quad e=1,2,...,E. \quad (2.4)$$

Below 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:

**instance**

set $X=\{x_1,x_2,...,x_p\}$ of $p=3q$ elements ($|X|=p$) and a family $C$ of $n$ 3-subsets of $X$ ($C=\{C_1,C_2,...,C_n\}$, $C_i \subseteq X$, $|C_i|=3$, $i=1,2,...,n$), $n \geq q$

**question**

does $C$ contain a subfamily $C' \subseteq C$, of $q$ ($|C'|=q$) pair-wise disjoint subsets of $X$?

Consider the directed single-commodity flow graph depicted in Figure 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 $C_j$ is connected to vertex $x_i$ if, and only if, $x_i \in C_j$ (in the considered example $C_1=\{x_1, x_2, x_3\}$, $C_2=\{x_2, x_4, x_5\}$ and $C_n=\{x_{k_1}, x_{k_2}, x_{k_3}\}$). In the sequel we shall assume that $X=\cup C$, 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 $(s,C_i)$. 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 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 $(s,C_j)$ (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 Figure 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 $C_i$ 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 an ES-flow equal to $q+1$, or show that such a flow does not exist, in polynomial time, then we would give an answer to the X3C question in polynomial time. This proves that the following ES-flow problem (ESF):

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

is $NP$-complete.

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 $h$, equal to $q+1$, and the path structure and links’ capacities specified by the graph in Figure 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, 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 \( \text{NP} \)-complete.

\[
\text{Fig.3. An instance of the single-commodity flow graph}
\]

3. Optimisation tasks
Let us first notice that FAP can be formulated as a mixed linear-integer programme [6] for directed graphs, using the node-link formulation of the multicommodity problems. The given demand to be allocated from node \( v \) to node \( t \) is given by \( d(v,t) \). Below, \( o(e) \) and \( t(e) \) denote the starting and end nodes of link \( e \), respectively.

Let \( V \) denotes 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**

\[
\forall e \in E \forall v \in V \forall v \neq t \quad \sum_{o(e)=v} f(e,t) - \sum_{t(e)=v} f(e,t) = d(v,t) 
\]

(3.2)
\( \forall e \in E \quad \sum_{v \in V} f(e, t) \leq c(e) \) \hspace{1cm} (3.3)
\( \forall e \in E \quad 0 \leq f_i(o(e), t) - f(e, t) \leq (1 - \delta(e, t)) \sum_{v \in V} d(v, t) \) \hspace{1cm} (3.4)
\( \forall e \in E \quad f(e, t) \leq \delta(e, t) \sum_{v \in V} d_i(v, t) \) \hspace{1cm} (3.5)
\( \forall e \in E \quad 0 \leq W(t(e), t) + w(e) - W(o(e), t) \leq (1 - \delta(e, t)) \mathcal{V} \) \hspace{1cm} (3.6)
\( \forall e \in E \quad 1 - \delta(e, t) \leq (W(t(e), t) + w(e) - W(o(e), t)) \mathcal{V} \) \hspace{1cm} (3.7)
\( \forall e \in E \quad \sum_{v \in V} \delta(e, t) \geq 1. \) \hspace{1cm} (3.8)

**Remark:** Putting equality in the last constraint, we force the shortest paths to be unique.

Unfortunately, it turns out that the above MIP problem is very difficult to solve already for small networks, even by CPLEX [7], so we do not report any numerical results in for the above formulation in the paper.

FAP can be extended by adding an objective function. Below we present three such extensions.

**Task AT1 (residual capacity maximisation):**
additional constants:
- \( b_e \) value of one idle capacity unit of link \( e \)
objective: maximise \( C_1 = \sum_e b_e \times (y_e - \sum_d \sum_j a_{ej} x_{dj}(w)) \) \hspace{1cm} (3.9)
subject to constraints: (2.1), (2.2) and (2.3).

**Task AT2 (throughput maximisation):**
additional constants:
- \( H_d \) maximal allowable volume of demand \( d \)
- \( c_d \) value of one realised demand volume unit of demand \( d \)
additional variables:
- \( h_d \) actually realised volume of demand \( d \) (continuous variable)
objective: maximise \( C_2 = \sum_d c_d \times (\sum h_d - h_d) \) \hspace{1cm} (3.10)
constraints: (2.1), (2.2), (2.3) and
\( h_d \leq h_d \leq H_d \quad d = 1, 2, \ldots, D. \) \hspace{1cm} (3.11)

Objective function can also be used as a penalty function, instead of constraint (2.2):

**Task AT3 (allocation with penalty function):**
objective: minimise \( C_4 = \sum_e f_i(o(e), t) - f(e, t) \) \hspace{1cm} (3.12)
subject to: (2.1) and (2.4).

In [2], piece-wise convex linear functions (depending on \( y_e \)) are used for \( f_i(\cdot) \). It is stated in [2] that AT3 is also an NP-complete problem.

The objective function (2.1) in AT may either be skipped (then \( C(w) = 0 \) and we arrive at AT0 - the pure allocation task) or it can express a second order objective of the allocation task. For the latter case the following versions of AT can be considered.

**4. Direct approach**
As shown in Section 2, the allocation tasks considered in this paper (FAP, AT1, AT2) are NP-complete. Also, the branch and bound method of CPLEX is not effective for the MIP formula-
tions of the tasks (cf. beginning of Section 3). Hence, the heuristic methods are called for. In this section we present three heuristic algorithms for the solving the allocation tasks, based on Local Search, Simulated Annealing and Lagrangean Relaxation.

4.1. Weights' Adjustment (WA)
Below we describe a local search method that tries to directly compute a feasible link weight system. The method is based on an iterative adjustment of the link weights on the basis of the links’ current loads with respect to their (fixed) capacities. The algorithm increases the weights of overloaded links and reduces the weights of underloaded links. The algorithm handles one of the two following types of weight systems: (2.4) and
\[ 0.05 \leq w_e \leq K \quad \text{and} \quad w_e - \text{real}, \quad e=1,2,\ldots,E. \] (4.1)

The method is based on local search procedures:
- **Weight adjustment**: When the network is overloaded (there is at least one overloaded link) the procedure increases weights of overloaded links, and decreases weights for underloaded links.
- **Load optimisation**: When the network is underloaded exactly two links are selected: the most and the least loaded ones, and their weights are adjusted to achieve the highest network utilization.

The algorithm works iteratively according to the following pseudo-code.

```plaintext
begin
  for e:= 1 to E do set_counter(e); generate_initial_weight_system(w_old);
  for step:= 1 to max_step do
    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
        if link e is underloaded then counter(e):= counter(e)-1;
        if (link e is overloaded) or ((link e is underloaded) and (counter(e) < 0)) then
          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);
            w_old(e):= w_new(e);
            if (link e is underloaded) and (counter(e) < 0) then set_counter(e)
          end
    end for;
    route_demands(w_new);
    if network is underloaded then
      for e:=1 to E do S:= select_links_procedure(e);
      for e in S do
        w_new(e):= modify_weight(w_old(e),load(e));
        cost_old(e):= cost_new(e)
      end for;
    route_demands(w_new);
    for e:= 1 to E do cost_new(e):= compute_cost(e);
    for e:=1 to E do acceptation_examination(cost_new(e), cost_old(e));
    if rejection then modify_weight(w_new(e),load(e))
  end for;
end
```

7
The algorithm starts from independently, randomly generated link weights $w$ (according to (2.4) or to (4.1)). If the network is overloaded the following process is started.

All demands are routed according to the initial weight system, and the current cost of each link (cost_new($e$)) is calculated as follows. If link $e$ is underloaded, its cost is made equal to its current load (occupied capacity) $y_e$, i.e. cost_new($e$)=$y_e$. If the link is overloaded, its cost is a sum of the two terms: one equal to the capacity $y_e$, and a second equal to the square of the link load minus its capacity, i.e. cost_new($e$)=$y_e+(y_e-y_e)^2$ (the second term makes the algorithm react strongly to the link overload).

If a link is overloaded, its weight is increased in order to attempt to take away some demand flows from it. 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($e$).

The weight of an underloaded link is adjusted only at every several iterations. Our tests show that if we update the weights of underloaded links too frequently (e.g. at each iteration), disadvantageous cycles can occur, disturbing the convergence of the algorithm. To overcome this difficulty we use the following procedure. Each link $e$ is assigned a randomly initialised attribute - counter($e$). If link $e$ is underloaded, the value of counter($e$) is decremented at each iteration, and the link weight updated only if it becomes overloaded or if counter($e$) becomes negative. When the link weight is adjusted due to the underload condition, counter($e$) is reset randomly. The random setting is another factor helping to avoid cycles in the optimisation. At each iteration the algorithm evaluates the current weight system with respect to (i) minimisation of the number of overloaded links, and (ii) minimisation of the magnitude of the average link overload.

If the algorithm finds a weight system at which the network is underloaded, the second part of the algorithm is started. In this case the goal of the optimization is changed. The three following alternatives can be selected: maximising the average free capacity in the network, maximising the total free capacity in the network or minimising the variance of the free capacity volumes on the links.

At this optimisation phase the most loaded and the least loaded links will be selected and their weights will be increased and decreased by 1, respectively. Then all demands will be routed according to the new weight system, and the utilisation and performance parameters of the network will be calculated. If the utilisation of network has increased (i.e. the algorithm has moved towards the dedicated goal) the optimisation is continued. If the utilisation has decreased, the algorithm uses stochastic acceptance criteria to decide whether the new weight system is acceptable or not. This criteria are very similar to that used in the Simulated Annealing process (cf. Section 4.2). Briefly, a new weight system, which results in worse network utilisation can be accepted at the beginning of the load optimisation process, but will be rejected after about 5-10% of the total number of steps. If the current solution is better than all the solutions found so far, the algorithm will store the current weight system, so it can be used as a final solution at the end of whole optimization process.

It is possible that accepting a worse weight system results in an overloaded network. In this case the above described weight adjustment method will be used again. During the whole optimisation process, the weight adjustment and the load optimization processes work iteratively, according to the current network state (over- or underloaded network).

4.2. Simulated Annealing (SAN)

Simulated Annealing is a well known multi-purpose meta-heuristic for combinatorial optimisation (cf.[8]). In many cases the approach 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, easy implementation, and sometimes relatively short running times. For the network design purpose it has been applied e.g. in [9].

In our implementation the integer weight systems (2.4) are assumed.

```plaintext
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 do
    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 with 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 current cost difference between the states ($ΔC$). At the beginning, SAN will accept states with relatively large $ΔC$ with a high probability, which is 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: $C(w)=\sum e \max\{\sum_d \sum_j a_{ej} x_{dj}(w) - y_e,0\}$.

### 4.3. Lagrangean Relaxation (LR)

Consider the following linear programming optimisation task:

**Allocation Task AT4 (non-constrained allocation)**

**objective:** maximise $C_4 = \sum_e b_e \times (y_e - \sum_d \sum_j a_{ej} x_{dj})$  

**constraints:**

\[ \sum_d x_{dj} = h_d \quad d=1,2,\ldots,D \]  
\[ \sum_d \sum_j a_{ej} x_{dj} \leq y_e \quad e=1,2,\ldots,E, \]

where $x_{dj} \geq 0$ is the flow realising demand $d$ allocated to path $j$. 

Using LR we solve the dual problem to AT4 (cf. [10]). The solution will yield a weight system that can be used for the ECMP routing. The dual problem is obtained by dualising (4.3) and (4.4), and forming the Lagrangean:

\[ L(\lambda, \lambda, x) = \sum_{e} b_{e}(y_{e} - \sum_{d} \sum_{j} a_{eq}x_{dj}) + \sum_{d} \lambda_{d}(h_{d} - \sum_{j} x_{dj}) + \sum_{d} \pi_{d}(\sum_{e} \sum_{j} a_{eq}x_{dj} - y_{e}) = \sum_{d} \lambda_{d}h_{d} - \sum_{e} (b_{e} + \pi_{e})y_{e} + \sum_{d} \sum_{j} \left( \sum_{e} a_{eq}(b_{e} + \pi_{e}) - \lambda_{d}\right)x_{dj}. \]  

(4.5)

**Dual Problem to AT4 (DP)**

maximise \( W(\pi, \lambda) = \min_{x} L(\pi, \lambda, x) \), over \( \pi \geq 0 \), and \( \lambda \) with unlimited sign. (4.6)

DP can also be solved with subgradient optimisation (cf. [11]) since it can be shown that (4.6) is equivalent to

\[ \text{maximise } V(\pi) = \sum_{e} (b_{e} + \pi_{e})(y_{e} - y_{e}) \text{ over } \pi \geq 0, \]  

(4.7)

where \( y_{e} \) 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 \( \pi \) the subgradient is calculated according to the formula:

\[ \frac{\partial V(\pi)}{\partial \pi_{e}} = y_{e} - y_{e}, \quad e = 1,2,\ldots,E. \]  

(4.8)

Alternatively, we can use the following formulation:

**Dual Linear Programming Problem (DLPP)**

maximise \( W(\pi, \lambda) = \sum_{d} \lambda_{d}h_{d} - \sum_{e} (b_{e} + \pi_{e})y_{e} \)

subject to

\[ \lambda_{d} \leq \sum_{j} a_{ed}(b_{e} + \pi_{e}), \quad j = 1,2,\ldots,m(d), \quad d = 1,2,\ldots,D \]  

(4.9)

\[ \pi, \lambda \geq 0. \]  

(4.10)

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 \( y_{e} \) for fixed \( \pi \)).

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:

\[ w_{e}^{0} = b_{e} + \pi_{e}^{0}, \quad e = 1,2,\ldots,E. \]  

(4.11)

The system \( w^{0} \) has a property that all the non-zero optimal primal flows (solving AT5) 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 AT1 of Section 2 (and the OSPF routing problem) is solved. However, the uniqueness of the shortest paths is not guaranteed (cf Section 6). In such a case we can anyhow use the weight system \( w^{0} \) for the ECMP routing. This in general will lead to a non-feasible solution to AT0, since the flows that solve the primal problem AT5 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.
5. Two-phase approach

As already pointed out, the OSPF allocation tasks formulated in Section 2 are not mathematical programming tasks and thus are difficult to solve in an exact way. Below we formulate a two-phase approach, in which both phases are based on mathematical programmes: Phase 1 on Mixed Linear-Integer Programming (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, a more complicated algorithm has 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 Phase 2 task.

5.1. Formulation of the two-phase optimisation task

**Phase 1 (MIP)**

**indices:**
- $d=1,2,...,D$ demands
- $j=0,1,...,m(d)$ paths for flows realising demand $d$
- $e=1,2,...,E$ links

**constants:**
- $h_d$ minimal volume of demand $d$
- $H_d$ maximal allowable volume of demand $d$
- $b_e$ value of one capacity unit of link $e$
- $c_d$ value of one demand volume unit of demand $d$
- $a_{edj}$ = 1 if $e$ belongs to path $j$ realising demand $d$, 0 otherwise
- $y_e$ capacity of link $e$
- $S$ large number

**variables:**
- $\varepsilon_{ej}$ binary variables forcing the single-path flow of demand $d$
- $x_{dj}$ flow realising demand $d$ on path $j$ (non-negative continuous variable)

**Variant A of Phase 1 (cf. AT1)**

**objective:** maximise $C_1 = \sum_e b_e \times (y_e - \sum_d (\sum_j a_{edj} \varepsilon_{ej}) h_d)$

**constraints:**
- $\sum_j \varepsilon_{ej} = 1 \quad d=1,2,...,D$ (5.2)
- $\sum_d (\sum_j a_{edj} \varepsilon_{ej}) h_d \leq y_e \quad e=1,2,...,E.$ (5.3)

**Variant B of Phase 1 (cf. AT2)**

**objective:** maximise $C_2 = \sum_d c_d \times (\sum_j x_{dj} - h_d)$

**constraints:** (5.2) and
- $\varepsilon_{edj} x_{dj} \leq c_d H_d \quad d=1,2,...,D, \ j=1,2,...,m(d)$ (5.5)
- $\sum_d \sum_j a_{edj} x_{dj} \leq y_e \quad e=1,2,...,E.$
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 \( d \)).

**Phase 2 (LP)**

**variables:** \( w_e \) weight of link \( e \) (continuous variable)

**constraints:**

\[
\sum_e a_{ed} w_e \leq 1 + \sum_e a_{ed} w_e \quad d=1,2,...,D, \quad j=1,2,...,m(d) \tag{5.6}
\]

\[
w_e \geq 1 \tag{5.7}
\]

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 \( j=0,1,...,m(d) \) should contain all the network paths for each demand \( d \), the above LP can be solved without generating all constraints (5.6), as explained in [3].

**5.2. Solving Phase 1**

The variants of Phase 1 are formulated as MIPs and can be effectively solved for small networks by means of MIP solvers, as CPLEX [7]. However, 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, the approximate heuristic methods can be effective in terms of solutions quality (sub-optimality) and of acceptable computation times. Below we describe two such approaches for solving Phase 1: Evolutionary Algorithm, and Simulated Allocation.

**5.2.1. Evolutionary Algorithm (EA)**

The use of Evolutionary Algorithms 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.

```plaintext
begin
  n:= 0; initialise(P_0);
  while not stop_criterion do
    O_n:= s;
    for i:= 1 to \lambda do P_n:= O_n \cup crossover(P_i); for e\in O_n do mutate(e);
    P_{n+1}:= select_best(O_n \cup P_n);
    n:= n+1
  end while
end
```

EA works with full allocation states \( \varepsilon=(\varepsilon_{dj}, \quad d=1,2,...,D, \quad j=0,1,...,m(d)) \), called chromosomes, satisfying constraints (4.2). Within a chromosome, each subsequence \( \varepsilon_d=(\varepsilon_{dj}, \quad j=0,1,...,m(d)) \) is called a gene (corresponding to demand \( d \)). Constraints (4.3) are taken into account via a penalty function, as will be shown below.

The algorithm starts with forming an initial population \( P_0 \) of \( \mu \) chromosomes, each generated randomly, with all genes satisfying constraints (4.2). At each step \( n \), a set \( O_n \) of \( \lambda \) 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

\[
C(\varepsilon) = \sum_e (\max\{\sum_d (\sum_j a_{ej} \varepsilon_{dj})h_d - y_e, 0\})^2. \tag{5.8}
\]
Having fixed the parents, their off-spring is formed by taking gene by gene at random from the parents (gene \( e_d, d=1,2,...,d \), is taken from a parent with probability 0.5).

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).

### 5.2.2. 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). Below we briefly describe an application of SAL to Variant A. In the application any set of currently used paths fulfills a simple necessary uniqueness condition illustrated in Figure 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 Figure 4.1 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 (cf.[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 exactly equal to the number of common links plus 1.

The SAL algorithm works with partial allocation flow sequences (states) \( \varepsilon=(\varepsilon_d, d=1,2,...,D, j=1,2,...,m(d)) \). The algorithm starts with the all zero-flow solution (\( \varepsilon_d=0 \)) and in each step chooses, with probability \( q(\varepsilon) \), between \( \text{allocate}(\varepsilon) \), i.e. adding one demand flow to the current flow sequence \( \varepsilon \), and \( \text{disconnect}(\varepsilon) \), i.e. removing one or more demand flows from the current solution \( \varepsilon \). We require that \( q(\varepsilon)>0.5 \), except for maximal allocation states \( \varepsilon \) for which \( q(\varepsilon)=0 \) (\( \varepsilon \) is a maximal allocation state if all demands are allocated: \( |\varepsilon|=\sum_d \sum_{j} \varepsilon_{dj} =D \)). Whenever a complete allocation state is reached, a check is made whether the cost of the best reached so far solution (\( \text{min\_cost} \)) is improved. Procedure \( \text{disconnect}(\varepsilon) \) is used in two variants:

- \( \text{disconnect\_1}(\varepsilon): \) remove from \( \varepsilon \) one previously allocated demand flow (at random)
- \( \text{disconnect\_2}(\varepsilon): \) remove from \( \varepsilon \) all the demand flows which use all the overloaded links, and, additionally, from a set of randomly, independently chosen links (with a certain probability).

```
begin
step:= 0; min_cost:= \infty; \varepsilon:= 0;
repeat
    step:= step+1;
    if random < q(\varepsilon) then allocate(\varepsilon) else if C(\varepsilon) < min_cost then disconnect\_1(\varepsilon) else disconnect\_2(\varepsilon);
    if \varepsilon \text{ is a maximal allocation state and } C(\varepsilon) < min\_cost then
        begin
            min\_cost:= C(\varepsilon); \varepsilon\_best:= \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 \( C(e)=\sum_a \max\{\sum_b (\sum_c (\sum_d a_{cd} e_{bd}) h_{e,v} y_{e}),0\} \) (expressing the total exceeded links’ capacity in the current allocation state) becomes greater than the current value of \( \min \text{cost} \).

Procedure allocate(e) assigns demand flow \( h_{e} \) to a selected path and increments the corresponding entry \( e_{a} \) 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 \( e \), the necessary feasibility condition described at the beginning of this section.

In fact, the allocation probability \( q(e) \) depends on the state through the number of allocated demands, i.e. \( q(e)=q(\lfloor e \rfloor) \). Of course, \( q(0)=1 \) and \( q(D)=0 \). One way to settle the allocation probabilities is to choose a threshold \( D \) (0<\( D \leq D \), e.g. \( D=0.8D \)), and to set \( q(k)=1 \) for \( k<\bar{D} \) and \( q(k)=q \) for \( \bar{D}<k<D \), for some fixed \( q>0.5 \).

The algorithm is terminated either when a feasible solution to the 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 1 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 \( s-c-...-d-f-...-a \) from node \( s \) in the tree under construction, and we consider labelling node \( b \) from node \( a \). Then we can label node \( b \) only when:

Case 1. Nodes \( a \) and \( b \) do not belong to a common path \( n(a,b)=0 \):

The path \( s-c-...-d-f-...-a \) in the currently constructed tree cannot contain any node \( v \) belonging to a common allocation path with node \( b \) \( n(v,b)=0 \) for \( v=s,c,...,d,f,...,a \).

Case 2. Nodes \( a \) and \( b \) belong to at least one common allocation path \( n(a,b)\geq0 \):

The path \( s-c-...-d-f-...-a \) in the currently constructed tree has the following property: nodes from \( s \) to \( d \) do not belong to a common allocation path with node \( b \), and \( f,...-a-b \) is a sub-path 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.

When node \( t \) is reached (this will always happen), 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 \( e \), the link metrics used in the labelling decisions are equal to \( 1+T\times\max\{0, y_{e}(e)+h_{e,v} y_{e}\} \), where \( y_{e}(e) \) is the load of link \( e \) in state \( e \) (i.e. \( y_{e}(e)=\sum_a (\sum_b (\sum_c (\sum_d a_{cd} e_{bd}) h_{e,v} y_{e}) \)) and \( T \) is a fixed, large positive number.

The above described labelling procedure works as 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 depicted in Figure 5
[6] depicts such a situation: all paths from \(s\) to \(t\) are inconsistent with the consistent set of paths \(\{p_1, p_2, ..., p_6\}\). 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.

![Diagram of a blocking situation in SAL](image)

**Fig.5. A blocking situation in SAL**

### 5.3. Solving Phase 2

The linear programme (4.11)-(4.12) 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. The following property holds:

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

**constants:**

\[
\begin{align*}
\sigma_{d_j} & \quad \text{non-negative continuous variable} \\
\text{variables:} & \\
\text{constraints:} & \\
\sigma_{d_j} \geq 0 & \\
\sum_{d} \sum_{w_{i \geq 0}} r_{edj} x_{ij} & \leq 0 \\
\sum_{d} \sum_{w_{i \geq 0}} \sigma_{d_j} & = 1
\end{align*}
\]

is infeasible.
6. Numerical results

Below we discuss applications of the approaches and algorithms described in Sections 3 and 4 for two network configurations: a 7-node (N7) and a 12-node (N12). N7 is an artificial network consisting of 7 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. Figure 6).

Individual demand volumes for N12 and N7 are given in Table 1.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>11</td>
<td>13</td>
<td>13</td>
<td>16</td>
<td>7</td>
<td>19</td>
<td>10</td>
<td>6</td>
<td>13</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3</td>
<td>21</td>
<td>19</td>
<td>21</td>
<td>14</td>
<td>32</td>
<td>14</td>
<td>5</td>
<td>68</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>27</td>
<td>21</td>
<td>25</td>
<td>17</td>
<td>35</td>
<td>14</td>
<td>3</td>
<td>107</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>21</td>
<td>15</td>
<td>51</td>
<td>21</td>
<td>19</td>
<td>84</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13</td>
<td>20</td>
<td>35</td>
<td>5</td>
<td>18</td>
<td>74</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>34</td>
<td>4</td>
<td>9</td>
<td>30</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>21</td>
<td>14</td>
<td>12</td>
<td>4</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>28</td>
<td>47</td>
<td>129</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>6</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>97</td>
<td></td>
</tr>
</tbody>
</table>

Tab.1. Demand volumes

In N7 and N12 bidirectional demands, links and paths are assumed. The directed (unidirectional) case will be treated afterwards, for the directed cases. 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: \(y(1,2)=75, y(1,6)=75, y(1,7)=79, y(2,3)=78, y(2,7)=87, y(3,4)=77, y(3,7)=97, y(4,5)=71, y(4,7)=100, y(5,6)=80, y(5,7)=95, y(6,7)=86\)

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 143MHz; LR and EA - Sparc II 250MHz; SAL - PC 366MHz) and the number of steps presented 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 much 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 reported in column 1 while the total links’ load exceeding the links’ capacities is shown in column 2. The three direct methods WA/C (continuous weights), WA/I (integral weights) and SAN, and the two-phase method SAL found the same final solutions in terms of the demand flows. For each network all demands were allocated to single paths (this is of course an intrinsic feature of SAL); 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, cf. Section 4.3).

<table>
<thead>
<tr>
<th>link</th>
<th>load</th>
<th>time(sec)</th>
<th>steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>LR</td>
<td>all</td>
<td>WA/C</td>
</tr>
<tr>
<td>N7-1</td>
<td>2</td>
<td>23</td>
<td>.18</td>
</tr>
<tr>
<td>N12-1</td>
<td>0</td>
<td>0</td>
<td>.24</td>
</tr>
<tr>
<td>N12-2</td>
<td>1</td>
<td>34</td>
<td>1.12</td>
</tr>
<tr>
<td>EA</td>
<td>SAL</td>
<td>EA</td>
<td>SAL</td>
</tr>
<tr>
<td>N7-1</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>N12-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>N12-2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Tab.2. Performance of the methods

For N12-1 the direct method LR found a nice weight system that implies unique shortest paths (identical to those found by all other methods considered in Table 2) and thus automatically 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 5 demands exactly 2 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 AT5 (cf. Section 3.3). The split is different from the ECMP split, and therefore the LR solution does not strictly solve the original problem AT0. 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 to that of WA, SAN and SAL. Unfortunately, this scheme does not work well for N7. Now, only 10 out of 21 demands have unique shortest paths, and there is a demand with as much as 7 shortest paths. In consequence, if we apply the ECMP flows for the LR weights then many links will be overloaded (and many underloaded) so the LR approach does not simply work (this is indicated with “-“ in columns 1 and 2). Despite these limitations, we note that LR, as a method applicable to large networks, may be used to yield initial weight systems (starting points) for other methods.

Finally, we have 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 to the one found with all other methods discussed above. 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 superscript “-“ 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 will indicate no feasible solution of Phase 1, then also the original problems AT0-AT4 will be most likely non-feasible.
For the above considered small networks, the WA method with integral weights (WA/I) is superior to the others in terms of the computation time; therefore WA/I can be recommended for small networks, the more that it is also simple. We have additionally tested WA/I by generating one hundred random weight systems for N12, routing the flows according to the ECMP rule for each system, and dimensioning the links by taking the links’ capacities equal to the resulting loads. In 95% of cases WA/I has been able to find weight systems that reproduced the same flows; for the remaining 5% of cases the solutions were very close to optimal, with typically only one link with exceeded capacity, and maximally with three such links.

7. Conclusions
In the paper we have formulated a set of OSPF related flow allocation problems and proposed a set of methods for solving them. We have shown that FAP, the basic OSPF flow allocation problem, is \(NP\)-complete. We have formulated a mixed linear-integer programming task for FAP; unfortunately the formulation turns out to be very difficult to solve even for such a sophisticated solver as CPLEX. Therefore, heuristic methods have to be applied for FAP and its extensions. Numerical studies indicate that the one-phase approach, consisting in direct finding feasible OSPF link weight systems with a Local Search method called Weights’ Adjustment, although very efficient for small networks, are quite effective even for large-scale networks with many, up to at least 112, nodes. We have also proposed a two-phase approach. In Phase 1 demands are allocated, using the Simulated Allocation heuristic, to single paths so that the resulting set of paths fulfills a necessary condition for the existence of a weight system, according to which these paths are unique shortest paths in the network graph. The desired weight system is then found in Phase 2 using Linear Programming.

References

18
