Heuristics for the MPLS Network Design with Single Path Minimum Weight Routing

Carlos Lopes, Amaro de Sousa
Institute of Telecommunications, University of Aveiro
3810-193 Aveiro, Portugal
clopes@av.it.pt, asou@det.ua.pt

***Abstract*— MPLS provides the flexibility required for managing the way the traffic is routed through the network. However, LSP configuring is a hard task in a growing network, as more and more routes need configuring and monitoring. A viable alternative is to rely on a minimum weight routing protocol to determine the paths used by the LSPs. An important requirement is that the routing paths should be unique to avoid the additional management complexity required by Equal Cost MultiPathing. Here, we address the dimensioning of MPLS networks with single path minimum weight routing. To solve large scale problem instances, we propose a methodology composed by two phases: first, the network design solution is computed based on a Greedy Randomized Adaptive Search Procedure (GRASP) with a neighbor structure given by an exponential growing difference set of values; then, the link weights are assigned solving an appropriate ILP model. For the network design phase, the proposed heuristic is compared with other heuristic strategies, namely, with another neighbor structure previously proposed and with a Simulated Annealing strategy. The computational results show a significantly better performance of the proposed heuristic.**

Keywords*— *MPLS network design, minimum weight routing, heuristics

I. INTRODUCTION

In the latest years, minimum weight routing technology has gathered a lot of interest by network operators mainly because of its ease of configuration and expansibility. IGP (Interior Gateway Protocol) routing protocols such as OSPF (Open Shortest Path First) and IS-IS (Intermediate System-Intermediate System) are the most used protocols in the today's IP networks. These protocols establish minimum weight paths through the network based on link weight assignment. The task of configuring and managing the way the traffic is routed through the network is based on assigning appropriate weights to the links. In the traditional IGP routing, when multiple paths with the same weight are present, the traffic is equally split among these paths (this is known as ECMP – Equal Cost MultiPathing).

MPLS (Multi-Protocol Label Switching) provides more flexibility than traditional IGP protocols by allowing for Traffic Engineering. Many IP service providers agree that MPLS Traffic Engineering can have beneficial impact on their networks [1-2]. A router that supports MPLS is known as a Label Switching Router (LSR). MPLS organizes the network in MPLS domains. The forwarding of IP packets from ingress to egress LSRs is done by means of routing paths, called Label Switched Paths (LSPs). In the ingress LSR, incoming IP packets are classified based on their destination and required quality of service and, depending on this classification, are forwarded through the appropriate LSP towards an egress LSR. In an MPLS domain, the ingress/egress nodes are designated as Edge LSRs.

While providing a lot of flexibility, MPLS can lead to undesirable micromanagement and complexity due to the large number of paths that grows ever larger with the size of the network. An alternative is to use MPLS combined with an IGP routing protocol such as OSPF or IS-IS in order to combine MPLS's flexibility with minimum weight routing's simplicity (see references [3] and [4]). With this approach, the LSPs are established based on minimum weight routing, thus freeing the operator of the task of configuring all the paths in the network. An important requirement is that the routing paths should be unique to avoid the additional management complexity required by ECMP.

II. RELATED WORK

In recent years, research has been focused on efficient weight assignment methods for traffic engineering purposes [3,4,6-16], i.e., a network configuration is considered with given link capacities and the weight assignment is exploited in order to optimize network performance. In [6] the authors address the problem of optimal routing in shortest path networks to minimize the average packet delay. They also enumerate and classify several types of routing problems used in networks as combinations of single-path routing, destination-based routing and shortest path routing. References [4] and [7] address the OSPF network routing task with the goal of minimizing the load on all links of the network considering that traffic flows can be routed based on ECMP mechanism. In these papers, algorithms based on local search heuristics are proposed for solving these problems. In [4] it is shown that OSPF routing leads to only a small decrease in

performance when compared to optimal routing. It is common practice for some network operators to forgo some performance in favor of a more easily configurable and manageable network.

In [8] and [9] genetic algorithms are used to solve OSPF networks optimization problems. In [8] the authors propose a genetic algorithm approach to solve the problem presented in [4]. In [9] the authors describe a method for solving a survivable version of this problem by determining the arc weights and multiplicities (number of links assigned to the arc) that accomplish the minimum total weighted multiplicity (multiplicity multiplied by the arc length) needed to route the required demand and handle any single arc or router failure. In our previous work [5], we consider that each arc of the graph has a multiplicity of 1 and that the capacities assigned to the arc can be treated as a single capacity, which is equal to the sum of these.

In [10], the authors prove the NP-Completeness of OSPF routing optimization tasks and discuss various heuristic approaches to solve these problems: Weight Adjustment Heuristics, Simulated Annealing, Lagrangean Relaxation, Simulated Allocation (see also [12]) and others. The authors formulate a two-phase approach to tackle these problems. They define phase I as the flow allocation task and phase II as the weight assignment task. In phase I they determine the single path flow allocation that optimizes a measure of the network performance such as residual link capacity. In phase II they determine a set of weights that provides the desired flow routing if such weights exist.

In [14] a Tabu-Search heuristic is used to solve the OSPF routing problem with unique shortest paths to determine link weights that provide the minimal utilization of the maximally used link. The authors also divide the problem in two phases. In phase I they solve the relaxed problem without the integrality of the weights and in phase II they use a Tabu-Search heuristic to find integer weights that can provide the routing configuration obtained in phase I.

The OSPF routing problem is solved in [15] with a branch-and-bound approach. This work considers ECMP and introduces a constraint that limits the number of outgoing links at a node to a maximum value, which may be suitable to operational use. In [16] the authors propose a branch-and-cut approach to solve the OSPF routing problem assuming also the ECMP rule.

Network design problems based on minimum weight routing protocols are also addressed in recent literature ([17-21]). For a comprehensive reference on modeling network design problems, please see [19]. Reference [20] presents a Lagrangean Relaxation heuristic for dimensioning OSPF networks with single minimum weight paths. This heuristic aims to determine the topology of the network, the hardware components and the link weights that minimize the total cost of the network. In reference [21] the authors present a MIP formulation of a survivable OSPF network design problem that includes link capacity and link weight determination to minimize a function based on cost, total capacity or capacity violation. To solve this problem, a few improvement based heuristics are also proposed. As in [20], traffic splitting is not considered. In [17], the authors present a MIP formulation and two heuristics for solving the design of OPSF networks with ECMP. This problem aims to determine simultaneously the link capacity assignment, the link metric assignment and the routing configuration.

Many of the strategies used to solve these problems are based on a two-phase approach. In the first phase a routing configuration is determined (and the link capacity, in the case of network design problems) and in the second phase the goal is to find a set of link weights that can provide the desired minimum weight paths. Depending on the method used for determining the routing configuration, this approach can sometimes lead to the infeasibility of the second phase. In some instances, the routing solution is not achievable via a minimum weight routing configuration. Destination based routing can often be accomplished by suitable choice of link weights, but that may not always be the case. For a study on the necessary and sufficient conditions to determine a minimum weight routing solution based on a destination based routing solution please see [22]. Reference [23] provides a polynomial method for determining routing instances that do not have a compatible set of link weights. In [24] the author formulates two problems for finding link weights that realize a prescribed set of routing paths as Inverse Shortest Path problems. The author shows how to compute approximate solutions via linear programming and shows that determining if a routing configuration can be implemented with a link weight system is NP-Hard. However, the author also states that for real world instances these problems are solved very efficiently with standard integer linear programming solvers.

III. SOLVING TECHNIQUES

Consider a network modeled by an undirected graph $G = (N,E)$ where node set N represents the set of LSRs and link set E represents non ordered pairs of LSRs that can be connected by transmission facilities. A transmission facility is characterized by a given cost to be put in operation at each link and a given bandwidth α that it provides on each arc of any link. The MPLS network design problem is the determination of the number of transmission facilities required in every link that can accommodate all traffic demands at the minimum cost. The traffic demands are defined by a set of traffic flows that are routed

according to a minimum weight routing protocol. The origin and the destination nodes belong to the set U which is the set of Edge LSRs. Since the traffic routing is supported by a single minimum weight routing path, we define for each non-ordered pair of origin and destination nodes p, q ($p \in U$, $q \in U \setminus \{p\}$) a single commodity with a demand value that is the sum of the demands of all the traffic flows between its end nodes.

The standard approach to implement heuristic algorithms for the design of networks with minimum weight routing is to assign different sets of link weights and, for each set, to determine the minimum weight routing paths and the required link capacities. This is straightforward when ECMP is considered since any set of link weights implicitly defines a network design solution. In the single path minimum weight routing case, this approach is not straightforward because a general set of link weights might contain multiple minimum weight paths for at least some pairs of network nodes. Moreover, there is no trivial way to determine a set of weights whose minimum weight paths are unique for all pairs of nodes. Therefore, to obtain a valid heuristic algorithm, we propose an approach with two phases: in the first phase, we determine a network design solution through the generation of sets of link weights; for each set, we select only one minimum weight path for each demand among the multiple ones that might be contained on it; in the second phase, we determine a suitable set of link weights defining the single minimum weight paths determined in the previous solution. In the following sub-sections, we will address separately each of these two phases.

A. First Phase Network Design

In the network design phase, for a given set of link weights, the minimum weight routing algorithm must carefully select the routing paths. In the cases where there is more than one minimum weight path, the paths selected by the minimum weight routing algorithm must have the following properties: (i) if the minimum weight path chosen from node p to node q goes through intermediate node i, this path should be the composed by the minimum weight path chosen from node p to node i plus the minimum weight path chosen from node i to node q; (ii) the minimum weight path chosen from node p to node q must cross the same nodes in inverse order than the minimum weight path chosen from node q to node p. In order to obtain a minimum weight routing algorithm with these properties, we propose a modification to the standard version of Dijktra's algorithm. At the beginning, all nodes are assigned an index, which is a positive integer value. Taking the standard Dijktra's algorithm for computing the minimum weight paths from a single origin to all other nodes, the following additional operations are included:

- Whenever a node is to be selected, if it has two predecessor nodes a and b giving the same weight, we compute, through the predecessors, the inverse paths from a and from b back to the first common node and, then, (i) we select the predecessor that gives the path with the minimum number of hops or (ii) if both paths have the same number of hops, we select the predecessor that gives the path containing the smallest index node.
- Whenever there are two nodes c and d to be selected with the same weight, we compute, through the predecessors, the inverse paths from c and from d back to the first common node and, then, (i) we select the node that gives the path with the minimum number of hops or (ii) if both paths have the same number of hops, we select the node that gives the path containing the smallest index node.

Now, any set of link weights has an associated network design solution that is computed in the following way: first, based on the proposed modified version of Dijktra's algorithm, we route the commodities through the minimum weight paths; then, we calculate the number of transmission facilities to install at each link to provide the required demand. The cost of this network design solution is the sum of the costs of the transmission facilities required on each link. For the first phase network design heuristic algorithm, we propose a network design search based on a Greedy Randomized Adaptive Search Procedure (GRASP) with the following neighbor structure: given a network solution associated to a set A of link weights, the neighbor set is composed by the network solutions associated to all valid sets B where B is different from A on a single link weight and this difference is one of the values of the difference set $X = \{1,-1,2,-2,4,-4,8,-8,16,-16\}$. We name this neighbor structure as the exponential growing difference set since it considers difference values that grow exponentially with the base of 2.

Note that X is composed by ten values, which means that the number of neighbors of a given network solution is equal to ten times the number of links. Note also that the small values of X might represent, in same cases at least, the same routing solution (there is no guarantee that the change of a link weight will change the minimum weight routing paths). The proposed difference set is a compromise between too many neighbors to compute and too many neighbors that represent the same network design solution.

The proposed heuristic algorithm is based on GRASP where for each random generated solution we run a local search procedure. In the local search phase of the algorithm, we search for the best solution among all neighbors and we let the algorithm move to the best neighbor solution even if it has the same cost value as

the present one. Nevertheless, we only allow a small number of consecutive moves for neighbors with the same cost. The algorithm stops when it reaches a predefined computing time. Consider *Wx* as the maximum positive value that can be assigned to the link weights (the minimum value is one). Given a set of link weights *A* (A_e is the weight value assigned to link $e \in E$, where $1 \leq A_e \leq Wx$), consider the function *Cost*(*A*) that computes the cost of the network design solution associated to it as described before. Consider also the parameter *MaxTime*, the algorithm total execution time, and *MaxCount*, the maximum number of consecutive moves allowed between neighbors with the same cost. In the algorithm description:

PresCost	is the cost of the network design solution associated to the present set of link weights *A*;
BestA	is the set of link weights that gives the best network design solution;
BestCost	is the cost of the best network design solution;
B	is the set of link weights of a neighbor of *A*;
BestB	is the set of link weights of the best neighbor of *A*;
BestNeigCost	is the cost of the network design solution associated to the best neighbor of *A*.

Algorithm OH1:

```
BestCost ← +∞;
Repeat:
    {
    Set A assigning weights for all links;
    PresCost ← Cost(A);
    count ← 0;
    Repeat:
        {
        If (PresCost < BestCost) do: { BestA ← A; BestCost ← PresCost; }
        BestNeigCost ← +∞;
        Sort the links in random order;
        For all links e ∈ E in the order sorted before do:
            For all values x ∈ X do:
                If (1 ≤ (Ae + x) ≤ Wx) do:
                    {
                    Set B equal to A and add x to the weight of link e in B;
                    If (Cost(B) ≤ BestNeigCost) do: { BestB ← B; BestNeigCost ← Cost(B); }
                    }
        If (BestNeigCost < PresCost) do: { A ← BestB; PresCost ← BestNeigCost; count ← 0; }
        Else if (BestNeigCost = PresCost) do: { A ← BestB; count ← count + 1; }
        Else do: count ← MaxCount;
        }
    Until (count = MaxCount)
    }
Until (MaxTime is reached)
```

In this algorithm, the outer Repeat cycle controls the execution time. At the beginning of this cycle, the link weight set *A* is randomly generated. Our computational experience showed that (i) small values of link weights usually obtain good results and (ii) assigning the weight value of 1 to all links frequently results in good solutions after the local search phase. Based on this experience, we have chosen to assign all the links with the weight value of 1 in the first execution of this cycle and to assign random values between 1 and 4 in the subsequent executions of this cycle. In the inner Repeat cycle, (i) the algorithm first stores the current solution if it is the best found so far, (ii) then, it computes the best neighbor solution among all possible neighbors (in the For cycles) and, finally, (iii) it moves to the best neighbor if its cost is better than the present solution (setting the *count* variable to zero) or if its cost is equal to the present solution (incrementing the *count* variable). Note that when the best neighbor has a worst cost, the *count* variable is set to *MaxCount*. Therefore, the inner Repeat cycle stops when the *count* variable is equal to *MaxCount*, which happens either if the best neighbor solution is worst than the present one or if the maximum number of consecutive moves allowed between neighbors with the same cost is reached. Note that before the computation of all neighbors (in the For cycles), the algorithm sorts the links in random order. Note also that when the best neighbor has the same cost of the current solution, the algorithm moves for the last equal cost neighbor computed in the chosen order. When there are multiple equal cost neighbors, the random sort prevents the algorithm from moving between the same pair of neighbors.

Recently, in [17], a Simulated Annealing algorithm was proposed for the variant of our network design problem that considers ECMP routing. In that work, the following neighbor structure was proposed: given a

network solution associated to a set A of link weights, the neighbor set is composed by the network solutions associated to all valid sets B where B is different from A on a single link weight and this difference is the minimum value that changes at least one minimum weight routing path. We name this neighbor structure as the minimum difference set. To understand how this minimum weight difference is computed, consider the following notation for a given traffic commodity k on graph G:

- w_k weight of the minimum weight path between the origin and destination of commodity k in graph G;
- w_{ke} weight of the minimum weight path between the origin and destination of commodity k in graph G without link e;
- w_{keo} weight of the minimum weight path between the origin of commodity k and closest node of link e;
- w_{ked} weight of the min. weight path between the destination of commodity k and closest node of link e.

For any link e, two minimum weight differences can be computed: x_{up} is the minimum positive value that, added to the weight of link e, will remove this link from at least one minimum weight path and x_{down} is the minimum positive value that, subtracted to the weight of link e, will add this link to at least one minimum weight path. These values are computed as follows:

$x_{up} \leftarrow +\infty$; $x_{down} \leftarrow +\infty$;
For all commodities k do:
  If (*link e is in the minimum weight path between origin and destination nodes of k*) *do*:
    $x \leftarrow w_{ke} - w_k + 1$;
    If ($x < x_{up}$) *do*: $x_{up} \leftarrow x$;
  Else do:
    $x \leftarrow (w_{keo} + w_{ked} + A_e) - w_{ke} + 1$;
    If ($x < x_{down}$) *do*: $x_{down} \leftarrow x$;

In this procedure, when link e is in the minimum weight path of commodity k, the difference x is the minimum difference that we have to add to the current link weight in order to take this link out of the minimum weight path. When link e is not in the minimum weight path of commodity k, the difference x is the minimum difference that we have to subtract to the current link weight in order to put this link in the minimum weight path. The procedure computes the differences for all commodities and it saves on each iteration the overall minimum values of x_{up} and x_{down}.

When comparing this minimum difference set with the previously explained exponential growing difference set, this has the merit of avoiding the search in neighbors with the same routing solution. However, the minimum difference set has two drawbacks. The first drawback is that it is computationally heavy. The second drawback is that it is conservative in the search space: it does not allow too many routing changes between neighbors. In the original work [17], this procedure does not sum one unit in the calculation of difference x since in their case, the ECMP can accept multiple paths with the same cost. The authors apply the minimum difference set to a Simulated Annealing algorithm where, a single link is randomly selected on each iteration. Given a set of link weights A, consider the function *UpperLimit*(A,e) that computes the minimum weight (the value $A_e + x_{up}$) higher than the one assigned to link e that changes at least one minimum weight path (when there is no such weight value, this function is equal to $+\infty$). Similarly, consider the function *LowerLimit*(A,e) that computes the maximum weight (the value $A_e - x_{down}$) lower than the one assigned to link e that changes at least one minimum weight path (when there is no such weight value, this function is equal to $-\infty$). Consider also the function *random*(a,b), with $a < b$, that computes a random value between a and b with an uniform distribution and the parameter β, a value between 0 and 1, which controls, on each iteration, the decay of the temperature parameter T.

The heuristic algorithm proposed in [17] and adapted to our network design problem is named OSA (presented in next page). In order to compare the performance of both algorithms, we adopted the same stopping criteria of OH1 algorithm. The algorithm stops when it reaches a predefined computing time that is controlled by the Repeat cycle. Before this cycle, an initial set A of link weights is randomly generated; the cost of its associated network design solution is assigned to the initial temperature T and saved as the cost of the best solution found so far. In the original work [17], the authors propose that the initial link weights be set to 50. However, our computational experience shows that initial weights of 1 usually achieve better results. In the Repeat cycle, (i) the algorithm first selects randomly a link, (ii) then, it computes the neighbor (whenever both increasing and decreasing minimum differences are valid weight values, the algorithm chooses randomly one of them with the same probability), (iii) then, it moves to the neighbor if it has a lower cost (and saves it if it is the best network design solution found so far) or moves to it with a probability of $e^{-Delta/T}$ if it has an higher cost and (iv) finally, it updates the temperature value T.

Algorithm OSA:

```
Set A assigning weights for all links;
PresCost ← Cost(A);
T ← PresCost;
BestA ← A;
BestCost ← PresCost;
Repeat:
    {
    Select randomly a link e ∈ E;
    If ((UpperLimit(A,e) ≤ Wx) or (LowerLimit(A,e) ≥ 1)) do:
        {
        If ((UpperLimit(A,e) ≤ Wx) and (LowerLimit(A,e) ≥ 1)) do:
            Assign randomly the value x either with UpperLimit(A,e) or with LowerLimit(A,e);
        Else if (UpperLimit(A,e) ≤ Wx) do:  x ← UpperLimit(A,e);
        Else do: x ← LowerLimit(A,e);
        Set B equal to A and assign x to the weight of link e in B;
        Delta ← Cost(B) – PresCost;
        If (Delta < 0) do:
            {
            A ← B;
            PresCost ← Cost(A);
            If (PresCost < BestCost) do: { BestA ← A; BestCost ← PresCost; }
            }
        Else if (Random(0,1) < e^(–Delta/T)) do: { A ← B; PresCost ← Cost(A); }
        T ← β × T;
        }
    }
Until (MaxTime is reached)
```

Algorithm OH2:

```
BestCost ← +∞;
Repeat:
    {
    Set A assigning weights for all links;
    PresCost ← Cost(A);
    count ← 0;
    Repeat:
        {
        If (PresCost < BestCost) do: { BestA ← A; BestCost ← PresCost; }
        BestNeigCost ← +∞;
        Sort the links in random order;
        For all links e ∈ E in the order sorted before do:
            {
            x ← UpperLimit(A,e);
            If (x ≤ Wx) do:
                {
                Set B equal to A and assign x to the weight of link e in B;
                If (Cost(B) ≤ BestNeigCost) do: { BestB ← B; BestNeigCost ← Cost(B); }
                }
            x ← LowerLimit(A,e);
            If (x ≥ 1) do:
                {
                Set B equal to A and assign x to the weight of link e in B;
                If (Cost(B) ≤ BestNeigCost) do: { BestB ← B; BestNeigCost ← Cost(B); }
                }
            }
        If (BestNeigCost < PresCost) do: { A ← BestB; PresCost ← BestNeigCost; count ← 0; }
        Else if (BestNeigCost = PresCost) do: { A ← BestB; count ← count + 1; }
        Else do: count ← MaxCount;
        }
    Until (count = MaxCount)
    }
Until (MaxTime is reached)
```

As will be seen in the following section, the OH1 algorithm has better performance than the OSA algorithm. To better understand the merits of OH1 algorithm, we have also implemented a second version,

named OH2, of the proposed GRASP based algorithm replacing the exponential growing difference set by the minimum difference set neighbor structure.

B. Second Phase Link Weight Assignment

The solution given by the first phase algorithm defines not only the required transmission facilities to be setup on each link but also the single minimum weight paths that must be observed through an appropriate set of link weights. As explained before, although these single minimum weight paths were computed based on a set of link weights, this set is not an appropriate solution since in the general case it contains multiple minimum weight paths. To solve this problem, we rely on an ILP (Integer Linear Programming) model and solve it through branch-and bound. Note that, in the general case, the solution given by the network design algorithm might have less links and less nodes than the ones defined on the original graph. The original links that are not included in any minimum weight path do not exist in the network design solution. Moreover, any original node that has no outgoing link does not exist also in the network design solution. Consider the graph $G = (N,E)$ where N is the set of network nodes that are in the first phase solution and set E is the set of links that also are in the first phase solution. Each link is designated by $\{i,j\}$ ($i \in N$, $j \in N \backslash \{i\}$) and is composed of arcs (i,j) and (j,i). Set A is the set of all arcs. To model the link weight assignment problem, we consider the binary parameters z_{ij}^q that when are one, indicate that arc (i,j) is in the minimum weight path from node i to node q on the first phase solution. We consider also the following variables: $w_{\max}$ is integer variable with the maximum weight value assigned to all arcs; w_{ij} is integer variable with the weight value to be assigned to arc (i,j) and π_i^q is the integer variable with the weight of the minimum weight path from i to q.

The link weight assignment problem aims to find a set of link weights minimizing the maximum weight value and is defined by the following ILP model:

Minimize $w_{\max}$

Subject to:

$$z_{ij}^q + w_{ij} + \pi_j^q - \pi_i^q \geq 1, \qquad \forall (i,j) \in A, \forall q \in U$$

$$z_{ij}^q + (w_{ij} + \pi_j^q - \pi_i^q)/M \leq 1, \qquad \forall (i,j) \in A, \forall q \in U$$

$$w_{ij} \geq 1, \qquad \forall \{i,j\} \in E$$

$$w_{ij} \leq w_{\max} \qquad \forall \{i,j\} \in E$$

$$w_{ij} = w_{ji}, \qquad \forall \{i,j\} \in E$$

$w_{\max}$, w_{ij} and π_i^q non negative integers

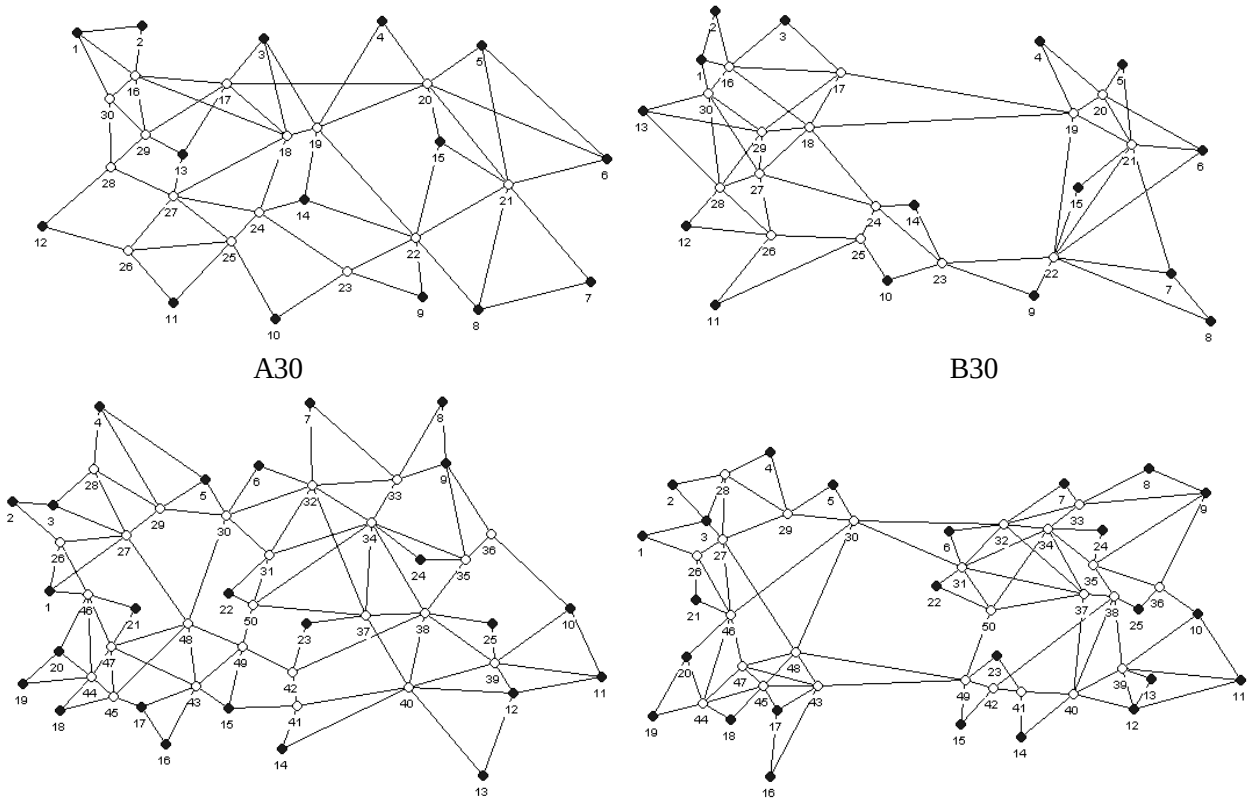
It is known [22] that in the general case solving this ILP model with branch-and-bound can be either computationally hard or unfeasible. A necessary but not sufficient condition to have a solution to this problem is that the routing paths should observe the destination based routing property: the superposition of all routing paths for the same destination node should form a directed tree towards that node. The first phase routing solution not only observes this property but also has the following additional property: there is a set of link weights where each routing path is one of the minimum routing paths defined by it. Although mathematical proof is required, our computational experience (next section) suggests that this additional property makes the solution of the link weight assignment problem always possible and easy to obtain.

IV. COMPUTATIONAL RESULTS AND CONCLUSIONS

The computational tests were conducted on four randomly generated Euclidian networks: A30, B30, A50 and B50 (Figure 1). Networks A30 and B30 have 30 nodes and 60 links, while A50 and B50 have 50 nodes and 100 links. Half of the nodes in all networks have been chosen to be edge LSRs (darker nodes on Figure 1). In all cases, we have considered the design of the networks based on SDH STM-4 transmission facilities with 622 Mbps of capacity. Transmission facility costs were modeled as a sum of two components: a fixed switching cost and a per unit length transmission cost. In the computational tests, the cost of each transmission facility link $\{i,j\}$ is given by $c_{\{ij\}} = 100 + 5 \times (\textit{length of link})$. Link lengths vary between 48 and 299 in network A30, 44 and 393 in network B30, 41 and 217 in network A50 and 37 and 256 in network B50.

Concerning the demand matrices, we have considered three different average demand values: (L)ow, (M)edium and (H)igh. For the networks with 30 nodes these values are: 75, 150 and 300 Mbps. For the networks with 50 nodes these values are: 50, 100 and 150 Mbps. This results in a total of 12 case studies. For all traffic matrices, the demands values were randomly generated for all pairs of edge LSR nodes with a uniform distribution between 20% and 180% of the average traffic demand.

The 12 case studies were solved by the three heuristic algorithms (OH1, OH2 and OSA) to determine a network design solution. For each obtained solution, the appropriate link weights were assigned by solving the ILP model with CPLEX. All computations were done on a 2GHz Pentium IV PC with 512MB RAM and the MS Windows operating system. For each algorithm, we set two different execution time values determined in the following way: we run OSA algorithm to determine how long it took to run around 5000 iterations and around 10000 iterations. The first value is designated as the Short Time while the second value is designated as the Long Time. For the networks with 30 nodes, the Short Time value was 45 seconds and the Long Time value was 90 seconds. For the networks with 50 nodes, the Short Time value was 5 minutes and the Long Time value was 10 minutes.



A30 B30

A50 B50

Figure 1: Case Studies network graphs

Since the heuristic algorithms are stochastic processes, different runs give different solutions. Therefore, for each of the execution time values, we run each algorithm 10 times. In the OSA algorithm, we set the parameter $\beta = 0.999$ for the Short Time runs and $\beta = 0.9995$ for the Long Time runs. Note that the correct tuning of this parameter is of great importance in the efficiency of the algorithm and one of the disadvantages of OSA algorithm when compared with the proposed OH1 and OH2 algorithms. These values were chosen in such a way that the probability of moving to a worst neighbor in the last iterations is neither too high nor too low. In the OH1 and OH2 algorithms, we set the parameter *MaxCount* = 10 (other values around the one chosen do not change significantly the efficiency of these algorithms). Figure 2 shows the average, minimum and maximum costs obtained in the 10 runs with the three network design algorithms for the case studies A30 and A50 (the results of case studies B30 and B50 are similar and, due to space constraints, are not presented here).

The obtained results show that the OH1 is always the best algorithm both in terms of average results and in terms of the best out of ten results. OH1 is on average 4% better than OSA, which is a significant improvement. Moreover, this gain is roughly the same for the Short Time runs and for the Long Time runs. The relative results of OH2 show that the merits of the proposed OH1 are in both the proposed heuristic strategy and in the proposed neighbor structure:

- OH2 is better than OSA, which means that for the same neighbor structure, a search procedure based on computing all neighbors produces better results than a search procedure that computes only a random selected neighbor.
- OH1 is better than OH2, which means that for the heuristic strategy, the neighbor structure based on the exponential growing difference set produces better results than the minimum difference set; this

result confirms the idea that although the minimum difference set avoids the search in neighbors with the same routing solution, the exponential growing difference set is more efficient in the computation of the neighbors and, since it allows the movement to neighbors with more routing differences, it can find better solutions.

Concerning the second phase link assignment problem, all network design solutions were solved by the ILP model using branch-and-bound. The average computing time to solve the ILP model for the case studies with 30 nodes was 0.17 seconds, while the worst case was solved in 2.7 seconds and the second worst case was solved in 2.3 seconds. For the case studies with 50 nodes, the average computing time was 11 seconds, the worst case was solved in 13 minutes and the second worst case was solved in 2.2 minutes. Note that for each case study, we run ten times three algorithms for two execution time values which gives a total of 60 network design solutions. Therefore, the ILP model was solved for 360 network design solutions of all case studies with 30 nodes and another 360 solutions of all case studies with 50 nodes. These computing times suggest that the minimum weight routing paths of the solutions found by the network design heuristic make the solution of the link weight assignment problem always possible and easy to obtain.

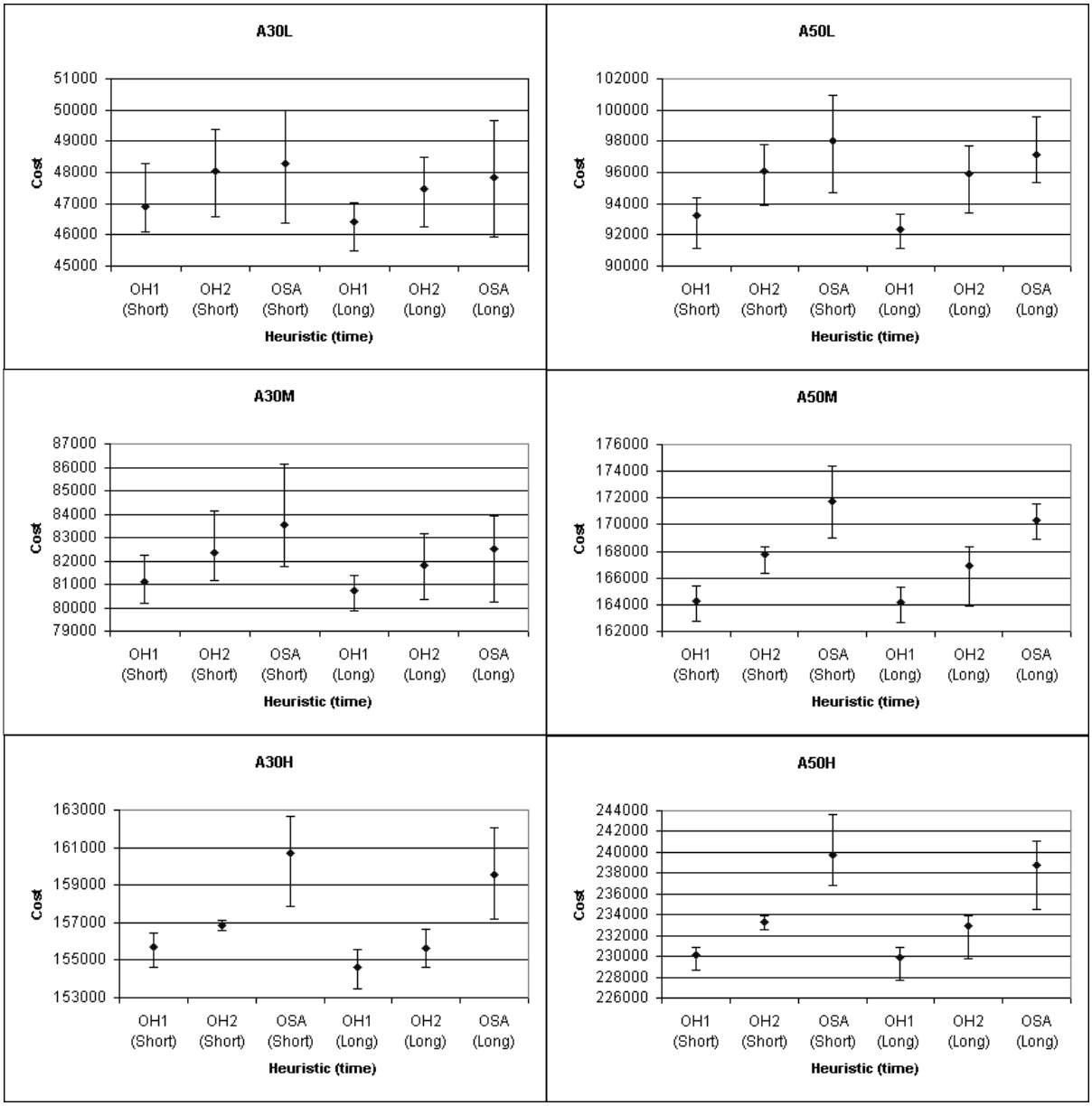


Figure 2: Cost results obtained for Case Studies A30 and A50

Finally, we have tried to find the optimal network design solution of case study A30L using the best performing ILP models proposed in [5]. This was done in order to motivate the need for the development of good heuristic algorithms addressing these problem sizes. We set the upper cutoff limit of branch-and-bound equal to the best cost found over all heuristic runs (in the A30L case study, the best network design solution was found with a cost of 43304). The Linear Relaxation value of this problem is 3,7676e+4. We used the

same PC platform used for the heuristics and, after 6 days of continuous run, the branch-and-bound procedure: (i) did not find any solution better or equal to 43304 and (ii) improved the initial lower bound value of 3,7676e+4 to a new lower bound value of 3,8342e+4, which is yet much lower than the best upper bound obtained by the heuristics. This illustrates that it remains very difficult to obtain optimality proved solutions for this type of problems addressing the instance sizes of the considered case studies.

V. ACKNOWLEDGEMENTS

The authors wish to acknowledge the financial support of FCT (Fundação para a Ciência e Tecnologia), Portugal, through grant SFRH/BD/6641/2001.

VI. REFERENCES

[1] D. Awduche, et al. J. Malcolm, J. Agogbua, M. O'Dell and J. McManus, "Requirements for Traffic Engineering Over MPLS," *IETF RFC 2702*, September 1999

[2] X. Xiao, A. Hannan, B. Bailey, and L. Ni, "Traffic Engineering with MPLS in the Internet", *IEEE Network*, March/April 2000

[3] Y. Wang, Z. Wang, L. Zhang, "Internet Traffic Engineering without Full Mesh Overlaying", *IEEE INFOCOM 2001*, April 2001

[4] B. Fortz, M. Thorup, "Increasing Intenet Capacity using Local Search", Technical Report, AT&T Labs Research, 2000. Preliminary short version of this paper published as "Internet Traffic Engineering by Optimizing OSPF Weights", *IEEE INFOCOM 2000*, March 2000

[5] C. Lopes, A. de Sousa, L. Gouveia, "Combined Link Dimensioning and Weight Assignment of Minimum Weight Routing Networks", NGI 2005, Rome, Italy, 18-20 April, 2005

[6] K. G. Ramakrishnan and M. A. Rodrigues, "Optimal routing in shortest-path data networks", *Bell Labs Technical Journal*, vol. 6, no. 1, pp. 117–138, 2001

[7] B. Fortz, M. Thorup, "Optimizing OSPF/IS-IS Weights in a Changing World", *IEEE Journal on Selected Areas in Communications*, vol. 20, No. 4, 2002

[8] M. Ericsson, M.G.C. Resende, P.M. Pardalos, "A Genetic Algorithm for the Weight Setting Problem in OSPF Routing", *Journal of Combinatorial Optimization*, vol. 6, pp. 299-333, 2002

[9] L. S. Buriol, M. G. C. Resende, M. Thorup, "Survivable IP Network Design with OSPF Routing", AT&T Labs Research Technical Report TD-64KUAW, September 6, 2004

[10] M. Pióro, Á Szentesi, J. Harmatos, A. Juttner, P. Gajowniczek and S. Kozdrowski, "On Open Shortest Path First Related Network Optimization Problems", *Performance Evaluation*, 42, pages 201-223, 2002

[11] A. Sridharan, R. Guerin, C. Diot, "Achieving Near-Optimal Traffic Engineering Solutions for Current OSPF/IS-IS Networks",*IEEE INFOCOM 2003*, April 2003

[12] P. Gajowniczek, M. Pióro, A. Szentesi, J. Harmatos, A. Jüttner, "Solving an ospf routing problem with simulated allocation," in *First Polish-German Teletraffic Symposium*, pp. 177–184, Dresden, 2000

[13] N. Bourquia, W. Ben Ameur, E. Gourdin, P. Tolla, "Optimal shortest path routing for internet networks," INOC 2003, pp. 119–125, Paris, 2003

[14] F. Ajili, R. Rodošek, A. Eremin, "A Scalable Tabu Search Algorithm for Optimising IGP Routing", INOC 2005, Book 2, pp. 348-354, Lisbon, 2005

[15] A. Eremin, F. Ajili, R. Rodošek, "A Set-based Approach to the Optimal IGP Weight Setting Problem", INOC 2005, Book 2, pp. 386-392, Lisbon, 2005

[16] A. Tomaszewski, M. Pióro, M. Dzida, M. Zagożdżon, "Optimization of Administrative Weights in IP Networks Using the Branch-And-Cut Approach", INOC 2005, Book 2, pp. 393-400, Lisbon, 2005

[17] Kaj. Holmberg, Di Yuan, "Optimization of Internet Protocol Network Design and Routing", *Networks*, vol 43(1), 39-53, 2004

[18] W. Ben-Ameur, E. Gourdin, B. Liau, N. Michel, "Designing Internet Networks", DRCN 2000, pages 56-61, Munich, April 2000

[19] M. Pióro and D. Medhi "Routing, Flow, and Capacity Design of Communication and Computer Networks, Morgan Kaufmann Publishers (Elsevier), 2004

[20] A. Bley., "A Lagrangian Approach for Integrated Network Design and Routing in IP Networks", INOC 2003, p. 107–113, Paris, 2003

[21] A. Bley, M. Grötschel, R. Wessäly, "Design of Broadband Virtual Private Networks: Model and Heuristics for the B-Win", Robust Communication Networks: Interconnection and Survivability, Volume 53 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, p. 1–16, 1998

[22] W. Ben Ameur, E. Gourdin, "Internet Routing and Related Topology Issues", *SIAM Journal on Discrete Mathematics*, vol. 17, no. 1, pp. 18-49, 2003

[23] P. Broström, K. Holmberg, "Determining the Non-Existence of a Compatible OSPF Metric", INOC 2005, Book 1, pp. 106-113, Lisbon, 2005

[24] A. Bley, "Finding Small Administrative Lengths for Shortest Path Routing", INOC 2005, Book 1, pp. 121-128, Lisbon, 2005