Quality of Service Constrained Routing Optimization using Evolutionary Computation

Miguel Rocha [a], Pedro Sousa [a], Paulo Cortez [b], Miguel Rio [c]

[a] *Center of Computer Science and Technology - CCTC, University of Minho*
Campus Gualtar, 4710-057 Braga, Portugal
Email: {mrocha,pns}@di.uminho.pt

[b] *Department of Information Systems / Algoritmi Center- University of Minho*
Campus Azurem, 4800-058 Guimarães, Portugal
Email: pcortez@dsi.uminho.pt

[c] *Department of Electronic and Electrical Engineering -University College London*
Torrington Place, WC1E 7JE, London, UK
Email: m.rio@ee.ucl.ac.uk

Abstract

In this work, a novel optimization framework is proposed that allows the improvement of Quality of Service levels in TCP/IP based networks, by configuring the routing weights of link-state protocols such as OSPF. Since this is a NP-hard problem, some algorithms from Evolutionary Computation were considered, working over a mathematical model that allows the definition of flexible cost functions that can take into account several measures of the network behaviour, such as network congestion and end-to-end delays. A number of experiments were performed, over a large set of network topologies, where Evolutionary Algorithms (EAs), Differential Evolution, local search methods and common heuristics were compared. EAs make the most promising alternative leading to solutions with an effective network performance, even under unfavourable scenarios. A number of state of the art multiobjective optimization algorithms were also tested, but the proposed EAs still hold as the most consistent method for network optimization.

Key words: Traffic engineering, Quality of Service, TCP/IP networks, OSPF, Evolutionary Algorithms

1 Introduction

In the last few years, several new types of applications have been integrated into IP based networks, fostering the development of novel network solutions that aim to provide end-users with Quality of Service (QoS) support. To accomplish this aim, distinct QoS aware architectures and specific traffic control mechanisms were proposed by the networking research community to provide distinct service levels to networked applications [28].

In the context of a QoS aware networking domain, Internet Service Providers (ISPs) have Service Level Agreements (SLAs) [16] with their clients and with peered ISPs that have to be strictly obeyed to avoid financial penalties. To successfully face such requirements, there is an important set of configuration tasks that have to be performed by administrators to assure that correct resource provisioning is achieved in the ISP domain. As an example, these configuration tasks may vary according with specific QoS provisioning solutions adopted by ISPs, and might include distinct tasks such as: specifying the forwarding treatment given at each network node (e.g. the per-hop-behaviours [15,8] of DiffServ [3]), defining traffic classification and packet marking rules at network edges, configuring admission control based mechanisms [27], configuring routing algorithms [23] or, in the case of finer-grain QoS solutions, enabling signaling protocols that allow per-flow resource reservation [5]. In this way, there is not a unique solution to create a QoS aware network infrastructure and, in general, any solution requires a number of components working together.

Independently of the set of mechanisms that might be in place in any QoS capable infrastructure, there are some components which have a crucial importance. One of such components has the ability to control the data path followed by packets traversing a given domain. There are two major alternative strategies for this purpose: Intra-domain routing protocols or Multi-Protocol Label Switching (MPLS) [9][2]. The use of MPLS presents significant drawbacks when used in the context of packet switching: firstly, it adds significant complexity to the IP model when compared with the simplicity of some routing protocols, since per-flow state has to be stored in every router of the path; secondly, MPLS Failure recovery mechanisms are considerably more complex than the typical router convergence ones; finally, it represents a considerable network management overhead.

The most commonly used intra-domain routing protocol is Open Shortest Path First (OSPF) [21][26]. Here, the administrator assigns weights to each link in the network, which are then used to compute the best path from each source to each destination using the well known Dijkstra algorithm [11]. The results of this method are then used to compute the routing tables in each node.

Since, in OSPF, the weight setting process is the only way administrators can affect the network behaviour this choice is of crucial importance and may have a major impact in the network performance. Nevertheless, in practice, simple rules of thumb are typically used in this task, like setting the weights inversely proportional to the link capacity. This approach often leads to sub-optimal network resource utilization.

An ideal way to improve the process of OSPF weight setting is to implement traffic engineering, assuming that the administrator has access to a matrix representing traffic demands between each pair of nodes in the network. This was the approach taken by Fortz et al [14] where this task was viewed as an optimization problem, by defining a cost function that measures the network congestion. The same authors proved that this task is a NP-hard problem and proposed some local search heuristics that compared well with the MPLS model. Another approach to this problem was the use of Evolutionary Algorithms (EAs) to improve these results [13]. Both alternatives were also combined to create hybrid EAs including local search operators that were able to improve these results [6].

However, previous approaches did not accommodate delay based constraints that are crucial to implement QoS aware networking services. In this work, two optimization algorithms from Evolutionary Computation (EAs and Differential Evolution) were used to calculate link-state routing weights that optimize traffic congestion while simultaneously complying with specific delay requirements. Thus, the main contribution of this work is to provide a multi-constrained QoS aware optimization framework. To accomplish this, a mathematical model of the problem that accommodates both congestion and delay constraints was proposed and a bi-objective cost function was defined. This will be used by the optimization algorithms to reach the optimal OSPF weights for each network link. The EAs and DE are evaluated by resorting to a large number of problem instances, where they are compared to other approaches such as local search, common heuristics and multiobjective optimization algorithms. In the comparison, the EAs emerge as the best alternative for this task.

The framework proposed in this paper should be viewed as a traffic engineering tool which, while focusing only at the OSPF routing level, aims at improving the overall QoS performance of a given domain. This does not hinder that other complementary QoS aware mechanisms might be used by network administrators, either to improve the network performance or to provide more strict QoS guarantees. The key point is that, based on the experiments, networks using the proposed optimization algorithms are able to clearly outperform the QoS performance obtained by networks using common OSPF weight setting heuristics. This means that irrespective of the QoS solutions in place, the proposed framework will always be an add-on to improve the QoS level of network

domains using OSPF routing mechanisms.

2 Problem description

2.1 General Description

The main objective of the proposed optimization framework is to provide network administrators with efficient OSPF link configurations, taking into account the users demands, the topology and other features of a given network domain (see Figure 1). This work assumes that client demands are mapped into a matrix, summarizing, for each source/destination router pair, a given amount of bandwidth and end-to-end delay required to be supported by the network domain. For instance, there are several techniques on how to obtain traffic demand matrices [18][10] which provide estimations regarding the overall QoS requirements within a given network domain.

*** insert Figure 1 around here ***

As an illustrative example, consider the network scenario included in Figure 1 and consider an individual demand between two network nodes (X and Y). If this demand is expressed in terms of a target delay, then the OSPF weight setting process should be able to provide a data path with the minimum propagation delay between X and Y (PATH 2). However, if no delay requirements are considered, and the only constraint between X and Y is a given bandwidth requirement, e.g. 90Mbps, then the OSPF setting process should try to minimize the network congestion and assign OSPF weights to force a data path inducing the lowest level of losses in the traffic (PATH 1). Considering now that a given demand has simultaneously bandwidth and delay constraints, then it is expected that the OSPF weight setting process should try to find a data path representing a trade-off between the bandwidth and delay metrics. In addition, if one considers that, after studying the QoS demands of the network domain users, each router pair of the domain may have specific multiconstrained QoS requirements (i.e. congestion vs. delay demands), then it is easy to understand the complex nature of the problem of obtaining OSPF weights able to optimize the QoS levels of a given network domain.

2.2 Problem Formulation

The general routing problem [1], that underpins our work, represents routers and transmission links by a set of nodes (N) and a set of arcs (A) in a directed

graph $G = (N, A)$. In this model, c_a represents the capacity of each link $a \in A$. Additionally, a demand matrix D is available, where each element d_{st} represents the traffic demand between each pair of nodes s and t from N. Let us assume that, for each arc a, the variable $f_a^{(st)}$ represents how much of the traffic demand between s and t travels over arc a. The total load on each arc a (l_a) can be defined in the following way:

$$l_a = \sum_{(s,t) \in N \times N} f_a^{st} \tag{1}$$

while the link utilization rate u_a is given by:

$$u_a = \frac{l_a}{c_a} \tag{2}$$

It is then possible to define a congestion measure for each link ($\Phi_a = p(u_a)$) [14], where p is a penalty function p that has small penalties for values near 0. However, as the values approach the unity it becomes more expensive and exponentially penalizes values above 1:

$$p(x) = \begin{cases} x, & x \in [0, 1/3) \\ 3x - 2/3, & x \in [1/3, 2/3) \\ 10x - 16/3, & x \in [2/3, 9/10) \\ 70x - 178/3, & x \in [9/10, 1) \\ 500x - 1468/3, & x \in [1, 11/10) \\ 5000x - 16318/3, & x > 11/10 \end{cases} \tag{3}$$

Under this framework, it is possible to define a linear programming instance [14], where the purpose is to set the value of the variables f_a^{st} that minimize the following objective function:

$$\Phi = \sum_{a \in A} \Phi_a \tag{4}$$

subject to:

$$\sum_{u:(u,v) \in A} f_{(u,v)}^{(s,t)} - \sum_{u:(v,u) \in A} f_{(u,v)}^{(s,t)} = \begin{cases} -d_{st}, & \text{if } v = s \\ d_{st}, & \text{if } v = t \\ 0, & \text{otherwise,} \end{cases} \qquad v, s, t \in N \tag{5}$$

$$l_a = \sum_{(s,t)\in N\times N} f_a^{st}, \qquad a \in A \tag{6}$$

$$\Phi_a \geq l_a, \qquad a \in A \tag{7}$$

$$\Phi_a \geq 3l_a - 2/3c_a, \qquad a \in A \tag{8}$$

$$\Phi_a \geq 10l_a - 16/3c_a, \qquad a \in A \tag{9}$$

$$\Phi_a \geq 70l_a - 178/3c_a, \qquad a \in A \tag{10}$$

$$\Phi_a \geq 500l_a - 1468/3c_a, \qquad a \in A \tag{11}$$

$$\Phi_a \geq 5000l_a - 16318/3c_a, \qquad a \in A \tag{12}$$

$$f_a^{(s,t)} \geq 0, a \in A, s, t \in N \tag{13}$$

where d_{st} means the value of D in row s and column t. Constraints 5 represent flow conservation in the network, ensuring the desired traffic flow is routed from source s to destination t; constraints 6 define the way load is calculated on each arc; finally, constraints 7 to 12 define the penalties (cost) on each arc. In the following, the optimal solution to this problem is denoted by Φ_{Opt}.

In OSPF, all arcs are associated with an integer weight. Every node uses these weights in the Dijkstra algorithm [11] to calculate the shortest paths to all other nodes in the network, where each of these paths has a length equal to the sum of its arcs. All the traffic from a given source to a destination travels along the shortest path. If there are two or more paths with the same length, between a given source and a destination, traffic is evenly divided among the arcs in these paths (load balancing) [20].

Let us assume a given solution, i.e. a weight assignment (w), and the corresponding utilization rates on each arc (u_a). In this case, the total routing cost is expressed by:

$$\Phi(w) = \sum_{a\in A} \Phi_a(w) \tag{14}$$

for the loads and corresponding penalties ($\Phi_a(w)$) calculated based on the given OSPF weights. In this way, the OSPF weight setting problem is equivalent to finding the optimal weight values for each link (w_{opt}), in order to minimize the function $\Phi(w)$.

The congestion measure can be normalized over distinct topology scenarios, dividing by a scaling factor defined as:

$$\Phi_{UNCAP} = \sum_{(s,t)\in N\times N} d_{st}h_{st} \tag{15}$$

where h_{st} is the minimum hop count between nodes s and t.

Finally, the scaled congestion measure cost is defined as:

$$\Phi^*(w) = \frac{\Phi(w)}{\Phi_{UNCAP}} \tag{16}$$

and the following relationships hold [14]:

$$1 \leq \Phi^*_{OPT} \leq \Phi^*_(w_{opt}) \leq 5000 \tag{17}$$

It is important to note that when Φ^* equals 1, all loads are below 1/3 of the link capacity; in the case when all arcs are exactly full the value of Φ^* is 10 2/3. This value will be considered as a threshold that bounds the acceptable working region of the network.

To enable an enlarged set of QoS constraints, an extension to this model is proposed in this work. This enrichment allows the inclusion of delay requirements for each pair of routers in the network. These are modelled as a matrix DR, that for each pair of nodes $(s,t) \in N \times N$ (where $d_{st} > 0$) gives the delay target for traffic between origin s and destination t (denoted by DR_{st}). A cost function was developed to evaluate the delay compliance for each scenario (a set of OSPF weights). This function takes into account the average delay of the traffic between the two nodes (Del_{st}), a value calculated by considering all paths between s and t with minimum cost and averaging the delays in each.

It was considered that, in the scenarios where this work would be applicable, the delay in each path is dominated by the component given by propagation delays in its arcs and that queuing delays can be neglected. However, if required, queuing delays can be introduced in the model by approximating its values resorting to queuing theory [4], taking into account the following parameters at each node: the capacity of the corresponding output links, their utilization rates and more specific technical parameters such as the mean packet size and the overall queue size associated with each link.

The *delay compliance ratio* for a given pair $(s,t) \in N \times N$ is, therefore, defined as:

$$dc_{st} = \frac{Del_{st}}{DR_{st}} \tag{18}$$

A penalty for delay compliance can be calculated using function p. So, the γ_{st} function is defined according to the following equation:

$$\gamma_{st} = p(dc_{st}) \tag{19}$$

This, in turn, allows the definition of a delay minimization cost function, given a set of OSPF weights (w):

$$\gamma(w) = \sum_{(s,t)\in N\times N} \gamma_{st}(w) \tag{20}$$

where the $\gamma_{st}(w)$ values represent the delay penalties for each end-to-end path, given the routes determined by the OSPF weight set w.

This function can be normalized dividing the values by the sum of all minimum end-to-end delays (for each pair of nodes the minimum end-to-end delay $minDel_{st}$ is calculated as the delay of the path with minimum possible overall delay):

$$\gamma^*(w) = \frac{\gamma(w)}{\sum_{(s,t)\in N\times N} minDel_{st}} \tag{21}$$

It is now possible to define the optimization problem addressed in this work that is clearly multi-objective. Indeed, given a network represented by a graph $G = (N, A)$, a demand matrix D and a delay requirements matrix DR, the aim is to find the set of OSPF weights (w) that simultaneously minimizes the functions $\Phi^*(w)$ and $\gamma^*(w)$. When a single objective is considered the cost of a solution w is calculated using functions $\Phi^*(w)$ for congestion and $\gamma^*(w)$ for delays. For multi-objective optimization, all algorithms described in the following section use a linear weighting scheme where the cost of the solution is given by:

$$f(w) = \alpha\Phi^*(w) + (1-\alpha)\gamma^*(w), \alpha \in [0,1] \tag{22}$$

This scheme, although simple, can be effective since both cost functions are normalized in the same range. The parameter α can be used to tune the trade-off between both components of the cost function.

3 Algorithms for OSPF weight setting

3.1 Evolutionary Algorithms

Evolutionary Algorithms (EAs) [19] can address the problems defined in the previous section. In the proposed EA, each individual encodes a solution as a vector of integer values, where each value (gene) corresponds to the weight of a link (arc) in the network (the values range from 1 to w_{max}). Therefore, the size

of the individual equals the number of links in the network. The individuals in the initial population are randomly generated, with the arc weights taken from a uniform distribution.

In order to create new solutions, several reproduction operators were used, more specifically two mutation and one crossover operator:

- *Random Mutation*, replaces a given gene by a random value, within the allowed range;
- *Incremental/decremental Mutation*, replaces a given gene by the next or by the previous value (with equal probabilities) within the allowed range;
- *Uniform crossover*, a standard crossover operator [25], suitable when the order of the variables in the individual (solution) is not important. This operator works by taking two parents as input and generating two offspring. For each position in the genome, a binary variable is randomly generated: if its value is 1, the first offspring takes the gene from the first parent in that position, while the second offspring takes the gene from the second parent; if the random value is 0, the roles of the parents are reversed.

In the proposed EA, whenever a new individual needs to be created, one of the previous reproduction operators is selected, with equal probabilities. Unlike traditional EAs, only one operator is applied to create each new individual. In this context, the crossover internal probability is set to 1, meaning that when it is used the crossover will always be applied. The mutation operators always use an internal probability of 0.01, which means that when they are used to create offspring, they will modify in average 1% of the genes in the parent.

The overall structure of the EA is given by:

(1) Generate and evaluate the initial population (P_0).
(2) While the termination criteria is not met:
  (i) Select from P_t individuals for reproduction.
  (ii) Apply the reproduction operators to breed the offspring and evaluate them.
  (iii) Insert the offspring into the next population (P_{t+1}).
  (iv) Select the survivors from P_t to be kept in P_{t+1}.

The selection procedure is done by converting the fitness value into a linear ranking in the population, and then applying a roulette wheel scheme. In each generation, 50% of the individuals are kept from the previous generation, and 50% are bred by the application of the reproduction operators. In the experiments a population size of 100 individuals was considered.

3.2 Differential Evolution

The Differential Evolution (DE) method differs from the EA essentially in the reproduction operators. DE generates trial individuals by calculating vector differences between other randomly selected members of the population. A variant of the DE algorithm called DE/rand/1 was considered that uses a binomial crossover [24], so the following scheme is followed for each individual i:

(1) Randomly select 3 individuals r_1, r_2, r_3 distinct from i;
(2) Generate a trial vector based on: $\vec{t} = \vec{r}_1 + F \cdot (\vec{r}_2 - \vec{r}_3)$;
(3) Perform crossover between this vector and the vector of the current individual, with probability CR, using at least one dimension of the trial vector;
(4) Evaluate the candidate and use it in the new generation if it is at least as good as the current individual.

Since OSPF weights are integer, it is necessary to round the values used in the DE before the evaluation. It is important to notice that in the DE all individuals go through the previous reproduction step. In the experiments, the population size was 20, F was set to 0.5 and CR to 0.6.

3.3 Local search

A local search (LS) scheme was devised to improve the quality of a solution and works as follows: taking a set of weights w_i, a link is randomly selected to start the process. Firstly, it increases the value of this weight by 1, if this leads to a better solution. This process is repeated while the solution improves. If the first increase operation did not lead to a better solution, a decrease is tried and repeated while the solution improves. The process is repeated for the next position, until all positions have been tested. The overall process is then repeated while the solution improves.

Based on this LS operator, a multi-start LS (MS-LS) algorithm was devised: it starts with a random solution and applies the LS operator; this process is repeated and the best solution found is kept. The process is terminated when a maximum number of solutions have been evaluated.

A number of heuristic methods were implemented to assess the order of magnitude of the improvements obtained by the proposed methods when compared with the traditional weight setting heuristics:

- **InvCap** - sets each link weight to a value inversely proportional to its capacity;
- **L2** - sets each link weight to a value proportional to the its Euclidean distance;
- **Random** - a number of randomly generated solutions are analyzed and the best is selected.

4 Experiments and Results

4.1 Setup

To evaluate the proposed algorithms, a number of experiments were conducted. The experimental platform used in this work is presented in Figure 2.

*** insert Figure 2 around here ***

All the algorithms and the OSPF routing simulator were implemented using the Java language. A set of 12 networks was created using the Brite topology generator [17], varying the number of nodes ($N = 30, 50, 80, 100$) and the average degree of each node ($m = 2, 3, 4$). This resulted in networks ranging from 57 to 390 links. The link bandwidth was generated by a uniform distribution between 1 and 10 Gbits/s. The network was generated using the Barabasi-Albert model, using a heavy-tail distribution and an incremental grow type (parameters HS and LS were set to 1000 and 100, respectively).

Next, the demand and delay constraints matrices (D and DR) were generated. For each network, a set of three distinct D and DR matrices were created. A parameter (D_p) was considered, representing the expected mean of congestion in each link (values for D_p in the experiments were 0.1, 0.2 and 0.3). For DR matrices, the strategy was to calculate the average of the minimum possible delays, over all pairs of nodes. A parameter (DR_p) was considered, representing a multiplier applied to the previous value (values for DR_p were 3, 4 and 5). Overall, a set of $12 \times 3 \times 3 = 108$ instances of the optimization problem were considered.

The termination criteria was the same for all optimization algorithms (EAs, DE and MS-LS) consisting in the maximum number of solutions evaluated. This value ranged from 50000 to 300000, increasing linearly with the number of links in the instance. In all cases, w_{max} was set to 20. For all the stochastic algorithms, 10 runs were executed in each case.

The results are grouped into two sets according to the cost function used. The first considers a single objective cost function, for the optimization of network congestion. The latter considers the case of simultaneous optimization of congestion and delays. In all figures presented in this section, the data was plotted in a logarithmic scale, given the exponential nature of the penalty function adopted.

4.2 Congestion

Since the number of experiments is quite high, it was decided to show aggregate results that can be used to draw conclusions. Table 1 shows the results for all the networks, averaged by the demand levels (D_p), including in the last line the overall mean value. It is clear that the results get worse with the increase of D_p, as would be expected. Figure 3 plots the same results in a graphical way, showing in the white area the acceptable working region, i.e. the congestion requirements are satisfied, whereas an increasing level of grey is used to identify regions with increasing levels of service degradation.

The comparison between the methods shows a superiority of the EA that achieves solutions which manage a very reasonable behaviour in all scenarios. The heuristics manage very poorly, and even $InvCap$, quite used in practice, gets poor results when D_p is 0.2 or 0.3, which means that the optimization with the EAs assures good network behaviour in scenarios where demands are at least 200% larger than the ones where $InvCap$ would assure similar levels of congestion. The results of DE and MS-LS are acceptable, but nevertheless significantly worse than the ones obtained by the EA, and the gap increases with larger values of D_p.

Table 1
Results for the optimization of congestion (Φ^*) - averaged by demand levels

D_p	**Random**	**EA**	**DE**	**MS-LS**	**L2**	**InvCap**
0.1	75.8	1.02	1.02	1.12	216	1.50
0.2	499	1.18	1.41	1.50	772	57.7
0.3	893	1.73	3.64	6.08	1289	326
Overall	489	1.31	2.02	2.90	759	129

*** insert Figure 3 around here ***

Table 2 shows the results for congestion, averaged by the number of nodes in the network. It is clear that the results obtained by the EAs are quite scalable since the quality levels are not affected by the number of nodes in the network graph (the same is true when the analysis considers the number of links). The results obtained in this section show that the EA makes an effective method for the optimization of OSPF weights, in order to minimize the congestion of the network. These results confirm the findings of Ericsson et al [13], although a precise comparison of the approaches is impossible since the original data is not available.

Table 2
Results for the optimization of congestion(Φ^*) - averaged results by the number of nodes.

Nodes	**L2**	**InvCap**	**Random**	**EA**	**DE**	**MS-LS**
30	598	95.7	264	1.29	1.97	4.93
50	815	105	419	1.28	1.59	2.44
80	731	157	594	1.31	2.67	2.01
100	891	156	680	1.36	1.86	2.23

4.3 Simultaneous optimization of congestion and delays

In this section, the results for the simultaneous optimization of congestion and delays, using a linear weight combination of the objectives are discussed. The results are presented in terms of the values for the two objective functions (Φ^* and γ^*), since the value of f for these solutions can be easily obtained and is not relevant to the analysis.

In a first stage, only the value of $\alpha = 0.5$ will be considered, thus considering each aim to be of equal importance. Table 3 shows the results averaged by the demand level (D_p) for all the algorithms. From the table, it is clear that the EA outperforms all other algorithms, followed by the DE and MS-LS. The heuristics behave quite badly, when both aims are taken into account. Since, in the heuristic methods, the solution is built disregarding the cost function, the results for multi-objective optimization only change due to the increase in the D_p parameter. A similar picture is found looking at Table 4, where the results are averaged by the delay requirement parameter DR_p.

A different view is offered by Figure 4 where the results are plotted with the two objectives in each axis. In the left (a), the results averaged by D_p are shown, while in the right (b) those are averaged by DR_p. In these graphs, the good overall network behaviour of the solutions provided by the EA is clearly visible, both in absolute terms, regarding the network behaviour in terms of congestion and delays, and when compared to all other alternative methods.

Table 3
Results for the simultaneous optimization of congestion and delays - averaged by D_p, with $\alpha = 0.5$

D	**Random**		**EA**		**DE**		**MS-LS**		**L2**		**InvCap**	
	Φ^*	γ^*	Φ^*	γ^*	Φ^*	γ^*	Φ^*	γ^*	Φ^*	γ^*	Φ^*	γ^*
0.1	88.0	107	1.17	1.92	1.18	2.04	1.73	4.07	216	1.76	1.50	260
0.2	481	137	1.47	2.32	1.65	2.92	3.38	8.30	772	1.76	57.7	260
0.3	950	149	2.41	3.23	4.58	5.64	15.3	15.9	1289	1.76	326	260
All	506	131	1.68	2.49	2.47	3.53	6.81	9.44	759	1.76	126	260

Table 4
Results for the simultaneous optimization of congestion and delays - averaged by DR_p, with $\alpha = 0.5$

DR	**Random**		**EA**		**DE**		**MS-LS**		**L2**		**InvCap**	
	Φ^*	γ^*	Φ^*	γ^*	Φ^*	γ^*	Φ^*	γ^*	Φ^*	γ^*	Φ^*	γ^*
3	535	283	1.95	4.22	2.78	6.42	9.65	21.3	759	2.94	128	578
4	506	82.0	1.59	1.78	2.44	2.36	6.12	4.65	759	1.25	128	159
5	478	27.2	1.51	1.48	2.38	1.82	4.65	2.39	759	1.10	128	44.1

It is easy to see that no single heuristic is capable of acceptable results in both aims simultaneously. L2 behaves well in the delay minimization but fails completely in congestion; InvCap is better on congestion but fails completely in the delays. The DE gets results that are in an acceptable range, but are always significantly worse than those of the EAs, and MS-LS does not manage good results when the problem instances get harder.

*** insert Figure 4 around here ***

To study the impact of the parameter α, three distinct values were tested: 0.25, 0.5 and 0.75. The value of 0.5 considers each aim to be of equal importance, while the 0.25 favors the minimization of delays and the 0.75 will give more weight to congestion. In this analysis, only the DE and the EA (the two best algorithms in the previous case) will be considered. In Table 5, the results obtained were summarized averaging by the parameter α. The first column represents the parameter α; the next two indicate the results for the DE algorithm and, finally, the last two give the results of the EA for both congestion and delays, each with an extra information indicating the percentage by which this results exceed the ones obtained by the corresponding algorithm under a single objective cost function.

The results shown in this table make clear the effect of parameter α, once it

Table 5
Overall results for simultaneous optimization of congestion and delays - averaged by α

α	**DE**		**EA**	
	Φ^*	γ^*	Φ^* (%)	γ^* (%)
0.25	2.98 (46.5%)	2.81 (54.7%)	2.02 (47.2%)	2.33 (32.5%)
0.5	2.47 (23.4%)	3.53 (93.2%)	1.68 (25.7%)	2.49 (43.8%)
0.75	2.45 (18.6%)	5.08 (168.8%)	1.61 (19.5%)	2.92 (69.5%)

is possible to observe different trade-offs between the two objectives. Indeed, when α increases the results on congestion improve, while the reverse happens to the delay minimization. The EA provides smoother changes being able to offer better results with all configurations. In particular, the intermediate value of α (0.5) provides a good compromise between the two objectives. In this case, the overall results show that, in average, there is a 25% decrease in the congestion performance and around 44% in the delays minimization, both when comparing to single objective optimization. These values are quite good, since in this case both aims have to be simultaneously obeyed, even if they are contradictory. In fact, a decrease in the performance, when compared to single objective optimization would always be expected. If the absolute average values for both cost functions are taken into account this indicates a quite acceptable network performance, well within the defined working region.

Table 6 confirms the good scalability properties of both the EA and the DE. The results are almost constant for the different network sizes (in this case, measured by the number of nodes).

Table 6
Results for the simultaneous optimization of congestion and delays - averaged by the number of nodes ($\alpha = 0.5$).

Nodes	**DE**		**EA**	
	Φ^*	γ^*	Φ^*	γ^*
30	2.23	2.99	1.58	2.25
50	2.25	4.03	1.78	2.96
80	3.18	3.70	1.62	2.37
100	2.21	3.41	1.75	2.38

Finally, Figure 5 show graphs similar to Figure 4 but considering only the EAs and plotting the results for different values of α. The trade-offs between the two objectives are clear. Regarding the figure on the left (a), the obtained delay and congestion cost values are averaged for distinct values of traffic demands (D_p). Moreover, three distinct lines are plotted, each one representing the

results obtained assuming distinct values of α (0.25, 0.5 and 0.75). The results plotted in the figure on the right (b) show the obtained delay and congestion cost values averaged now for distinct values of the delay requests (DR_p). The results show the correctness of the proposed optimization model, since higher values of α lead to an improvement in the congestion metric but, at the same time, a decrease in the delay performance.

*** insert Figure 5 around here ***

5 Experiments with multiobjective optimization algorithms

The multiobjective optimization problem addressed in this work was approached in the previous sections by considering a linear weighting of the objective values of the two distinct optimization aims. This approach assumes that the parameter α can be set to fine tune the optimization process and achieve solutions with the adequate trade-off between the two targets. This strategy presented good results, but it suffers from one main drawback, since it assumes that there is one single trade-off that is optimum. Therefore, the algorithms typically return one single solution that has to be implemented by the network administrator.

An alternative would be to have algorithms that could return a set of solutions with distinct trade-offs between the two objectives, and let the network administrator decide which solution to implement. A number of algorithms have been proposed in the last few years to address this task, in the arena of Multiobjective Optimization (MOO) and EAs are among the most popular ones [7]. These methods are able to give as output not only one optimal solution, but rather a set of solutions that are non-dominated.

A solution is dominated by another solution, if the first is worse than the second in at least one of the objectives and it is not better in none. More precisely, the aim of these methods is to return a Pareto front (PF), i.e. a set of non-dominated solutions, for a given problem. This PF should be as near as possible to the optimal set of non-dominated solutions and also as distributed as possible, i.e. it should cover the whole set of possible trade-offs between the optimization aims.

In order to evaluate the performance of this class of algorithms, in the OSPF weight setting task previously defined, a number of state-of-the-art alternatives was implemented. In this task, the *jMetal* software [12] was used to implement the following algorithms: SPEA2, NSGA-II, PESA-II, PEAS, PSO, AbYSS and MOCell. These encompass several types of multiobjective EAs, namely Particle Swarm Optimization, Evolution Strategies and Scatter Search, thus

covering the most popular multiobjective optimization algorithms available today.

To conduct an evaluation of these methods, a subset of the previous network topologies (3 out of the 12) was considered (this reduction was imposed by the computational effort). All methods were run with the standard setup provided by the *jMetal* software and were stopped when a maximum number of solutions were evaluated as in the previous methods. The aim was to compare these algorithms among each other, but also to evaluate the merits of this approach versus the one described in the previous sections.

Evaluating the performance of MOO algorithms is a complex task and to compare the results of MOO approaches with traditional methods is still trickier. This study does not intend to be exhaustive in this comparison and two simple performance metrics were used to evaluate the approaches:

- *C-measure* [29]: It is based on the concept of solution dominance. Given two PFs ($PF1$,$PF2$), the measure $C(PF1, PF2)$ returns the fraction of solutions in $PF2$ that are dominated by at least one solution in $PF1$. A value of 1 indicates that all points in $PF2$ are dominated by points in $PF1$, so values near 1 clearly favour the method that generated $PF1$; values near 0 show that few solutions in $PF2$ are dominated by solutions in $PF1$. This concept can be extended to traditional single-solution methods by calculating $C(S1, PF2)$ where $S1$ is a single solution. Therefore, it simply indicates the proportion of solutions in $PF2$ that are dominated by $S1$.
- *Trade-off analysis (TOA)*: For a pareto front $PF1$, and given a value of β, the solution that maximizes $\beta\Phi^* + (1-\beta)\gamma^*$ is selected. Parameter β can take distinct values in the range $[0, 1]$, thus defining different trade-offs between the objectives (working in a way similar to the parameter α in previous sections, but applied only after the optimization process). The values with the same β can be compared among the several MOO algorithms and also with those from traditional algorithms. In this last case, only one solution is available, so the process is simplified.

In Table 7, the results for the C-measure are shown. The overall mean value for the distinct instances (27 in this case) and 10 runs was computed. For each instance, $C(M1, M2)$ is computed for all pairs of distinct runs of $M1$ and $M2$. In Table 8, the TOA results are displayed. As before, means were calculated over all instances and runs, for each value of β.

An analysis of the results of both tables shows that, when comparing the MOO approaches, the NSGA-II and SPEA2 outperform all other alternatives. In fact, in Table 7 they show the highest values in the rows and the lowest in the columns, thus having PFs with few dominated and many dominating solutions. In Table 8, they also present the lowest results for the distinct values

Table 7
Results for the C-measure in MOO (mean of $C(M1, M2)$ computed over all instances and runs).

	AbYSS	**MOCell**	**NSGA**	**PAES**	**PESA**	**PSO**	**SPEA**
AbYSS	0	0.350	0.028	0.112	0.163	0.099	0.018
MOCell	0.493	0	0.029	0.124	0.211	0.121	0.006
NSGA-II	0.921	0.905	0	0.456	0.707	0.650	0.347
PAES	0.651	0.631	0.224	0	0.451	0.328	0.167
PESA-II	0.680	0.600	0.165	0.225	0	0.343	0.120
PSO	0.773	0.734	0.122	0.260	0.419	0	0.070
SPEA2	0.889	0.866	0.420	0.418	0.740	0.632	0
EA-$\alpha = 0.25$	0.824	0.797	0.508	0.392	0.607	0.557	0.444
EA-$\alpha = 0.5$	0.781	0.785	0.461	0.360	0.560	0.572	0.416
EA-$\alpha = 0.75$	0.666	0.685	0.369	0.303	0.461	0.547	0.329

Table 8
Results for the TOA in MOO (mean over all instances and runs given the value of β).

Algorithm	$\beta = 0$	$\beta = 0.25$	$\beta = 0.5$	$\beta = 0.75$	$\beta = 1$
AbYSS	8.91	31.1	53.3	75.4	97.6
MOCell	8.12	33.9	59.7	85.5	111
NSGA-II	4.21	12.8	21.3	29.9	38.4
PAES	5.79	96.2	187	277	367
PESA-II	5.50	30.3	55.1	80.0	105
PSO	6.10	26.4	46.7	67.0	87.3
SPEA2	3.41	11.8	20.3	28.7	37.1
EA-$\alpha = 0.25$	3.21	2.91	2.60	2.30	1.99
EA-$\alpha = 0.5$	3.80	3.26	2.72	2.18	1.63
EA-$\alpha = 0.75$	4.19	3.52	2.84	2.16	1.48

of β. Overall, the SPEA2 seems to be the best alternative.

When comparing the MOO performance with the one obtained by the proposed EA, it is also clear that the solutions obtained by MOO are, in general, not good alternatives for network management (from Table 8). In fact, they are quite far from the results obtained by the EA that, regardless of the values of α (used in the optimization) and β, always shows quite low values. There-

fore, the proposed EA with linear weighting shows a better trade-off between both objectives, while MOO methods show a bias, behaving better in delay optimization and failing in congestion.

From Table 7 it is also possible to conclude that a large number of MOO solutions are dominated by the EA's solutions, while the reverse is not true. In fact, the columns for the EAs are not shown in the table because its values were always zero. This means that the final solution for the EA is never dominated by any solution obtained by a MOO algorithm.

6 Conclusions and further work

The optimization of OSPF weights brings important tools for traffic engineering, without demanding modifications on the basic network model. This work presented Evolutionary Computation approaches for multiobjective routing optimization in the Internet. Resorting to a set of network configurations, each constrained by bandwidth and delay requirements, it was shown that the proposed Evolutionary Algorithms (EAs) were able to provide OSPF weights that can lead to good network behaviour. The performance of EAs was compared with other algorithms (Differential Evolution, local search, heuristics) clearly showing its superiority.

Although a simple weighting method was used to face the multiobjective nature of the problem, the results were of high quality. This is probably due to the effort of normalizing both cost functions in a coherent manner. In fact, a comparison was conducted with a number of state-of-the-art methods for multiobjective optimization (MOO) and the results of the proposed EA still hold as the more consistent, presenting the best trade-off between both aims. This does not invalidate further research on the development of more effective MOO methods to search for a near-optimal pareto front, since returning a number of high quality solutions with distinct trade-offs between the two objectives is quite useful as an outcome to a network administrator.

The proposed optimization framework, although requiring some computational effort, can be achieved in useful time (e.g. for a network with 50 nodes, a good solution can be obtained by the EA in less than half an hour). If very distinct traffic profiles occur in different times of day (e.g. night and day) the corresponding matrices can be used to optimize distinct OSPF weights. Furthermore, the adaptation to a new solution is always faster than running from scratch, since a good solution is available to boost the search and therefore any change that demands reoptimization will be rapidly achieved. Given all these facts, we can say that the proposed framework could be implemented in a straightforward way in a real world scenario.

One important topic for future work is the integration of distinct classes of QoS demands in the proposed optimization model. On this topic, the Internet Engineering Task Force (IETF) has proposed standards on Multi-topology Routing aiming at providing different paths for different types of traffic [22].

Acknowledgments

The authors wish to thank the Portuguese National Conference of Rectors (CRUP)/British Council Portugal (B-53/05 grant), the Nuffield Foundation (NAL/001136/A grant), the Engineering and Physical Sciences Research Council (EP/522885 grant) and Project SeARCH (Services and Advanced Research Computing with HTC/HPC clusters), funded by FCT under contract CONC-REEQ/443/2001, for the computational resources made available.

References

[1] R.K. Ahuja, T.L. Magnati, and J.B. Orlin. *Network Flows*. Prentice Hall, 1993.

[2] D. Awduche and B. Jabbari. Internet traffic engineering using multi-protocol label switching (MPLS). *Computer Networks*, 40:111–129, 2002.

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. RFC 2475: An architecture for differentiated services, 1998.

[4] G. Bolch, S. Greiner, H. de Meer, and K. Trivedi. *Qeueing Networks and Markov Chains - Modeling and Performance Evaluation with Computer Science Applications*. Wiley-Interscience; 2 edition, 2006.

[5] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource Reservation Protocol (RSVP). *RFC 2205*, September 1997.

[6] L. Buriol, M. Resende, C. Ribeiro, and M. Thorup. A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. *Networks*, 46:36–56, 2005.

[7] C.A. Coello Coello. *Recent Trends in Evolutionary Multiobjective Optimization*, pages 7–32. Springer-Verlag, London, 2005.

[8] B. Davie, A. Charny, J. C. R. Bennet, K. Benson, J. Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. An Expedited Forwarding PHB (Per-Hop Behavior). *RFC 3246*, March 2002.

[9] B. Davie and Y. Rekhter. *MPLS: Multiprotocol Label Switching Technology and Applications*. Morgan Kaufmann, USA, 2000.

[10] A. Davy, D. Botvich, and B. Jennings. An efficient process for estimation of network demand for qos-aware ip networking planning. In G. Parr, D. Malone, and M. Foghlú, editors, 6^{th} *IEEE International Workshop on IP OPerations and Management, IPOM 2006, LNCS 4268*, pages 120–131. Springer-Verlag, 2006.

[11] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(269-271), 1959.

[12] J.J. Durillo, A.J. Nebro, F. Luna, B. Dorronsoro, and E. Alba. jMetal: A Java Framework for Developing Multi-Objective Optimization Metaheuristics. Technical Report ITI-2006-10, Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, E.T.S.I. Informática, Campus de Teatinos, December 2006.

[13] M. Ericsson, M.G.C. Resende, and P.M. Pardalos. A Genetic Algorithm for the Weight Setting Problem in OSPF Routing. *J. of Combinatorial Optimization*, 6:299–333, 2002.

[14] B. Fortz and M. Thorup. Internet Traffic Engineering by Optimizing OSPF Weights. In *Proceedings of IEEE INFOCOM*, pages 519–528, 2000.

[15] J. Heinenen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group. *RFC 2597*, June 1999.

[16] Y. Liu, C. Tham, and Y. Jiang. Conformance analysis in networks with service level agreements. *Computer Networks and ISDN Systems*, 47(6):885–906, 2005.

[17] A. Medina, A. Lakhina, I. Matta, and John Byers. BRITE: Universal Topology Generation from a User's Perspective. Technical Report 2001-003, http://citeseer.ist.psu.edu/article/medina01brite.html, January 2001.

[18] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic matriz estimation: Existing techniques and new directions. *Computer Communication Review*, 32(4):161–176, 2002.

[19] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, USA, third edition, 1996.

[20] J. Moy. *OSPF, Anatomy of an Internet Routing Protocol*. Addison Wesley, 1998.

[21] J. Moy. RFC 2328: OSPF version 2, April 1998.

[22] P. Psenak, S. Mirtorabi, A. Roy, L. Nguyen, and P. Pillay-Esnault. Multi-topology (mt) routing in ospf (internet draft), January 2006.

[23] T. Slattery and W. Burton. *Advanced IP Routing in Cisco Networks*. McGraw-Hill, 1999.

[24] R. Storn and K. Price. Differential Evolution - a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[25] Gilbert Syswerda. Schedule optimization using genetic algorithms. In L. Davis, editor, *Handbook of Genetic Algorithms*. Van Nostrand, 1991.

[26] T.M. ThomasII. *OSPF Network Design Solutions*. Cisco Press, 1998.

[27] T. Tsuchiya. Call Admission Control with QoS Class Modification. *IEICE Transactions on Communications, Special Issue on Internet Technology III*, E86-B(2):682–689, February 2003.

[28] Z. Wang. *Internet QoS: Architectures and Mechanisms for Quality of Service*. Morgan Kaufmann Publishers, 2001.

[29] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.

List of Figure Captions

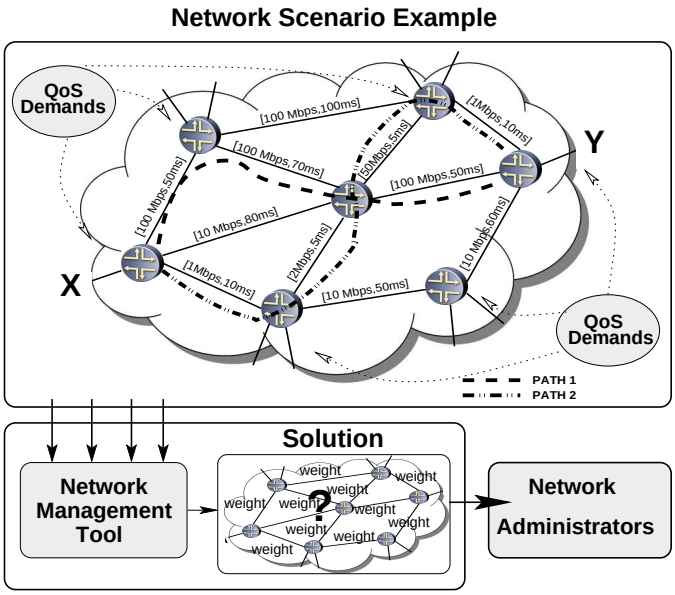

Fig. 1.

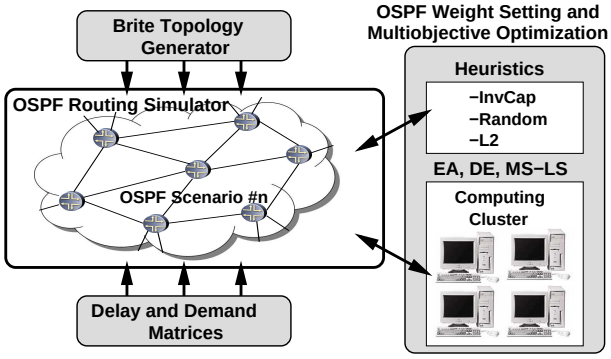

Fig. 2.

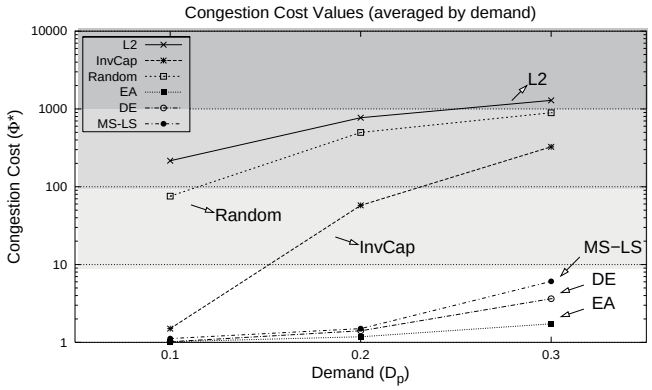

Fig. 3.

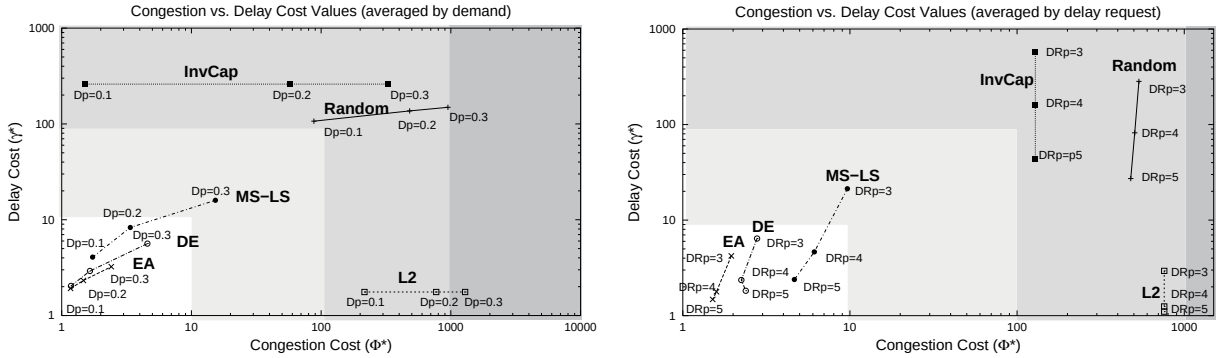
Congestion vs. Delay Cost Values (averaged by demand)
Delay Cost (γ*)
Congestion Cost (Φ*)
InvCap
Random
MS−LS
DE
EA
L2
Dp=0.1
Dp=0.2
Dp=0.3
Congestion vs. Delay Cost Values (averaged by delay request)
DRp=3
DRp=4
DRp=5
DRp=p5

Fig. 4.

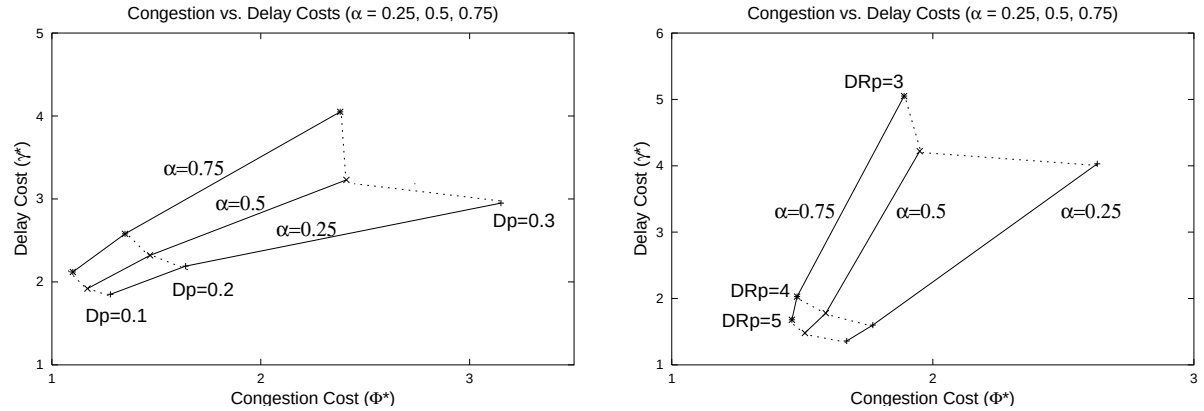
Congestion vs. Delay Costs (α = 0.25, 0.5, 0.75)
Delay Cost (γ*)
Congestion Cost (Φ*)
α=0.75
α=0.5
α=0.25
Dp=0.1
Dp=0.2
Dp=0.3
DRp=3
DRp=4
DRp=5

Fig. 5.