Comparative Study of Inter-Domain Traffic Optimization Algorithms

Manuel Pedro[1,2], Edmundo Monteiro[2], and Fernando Boavida[2]

[1]Polytechnic Institute of Leiria
ESTG, Morro do Lena – Alto do Vieiro
2411-901 Leiria
PORTUGAL

[2]University of Coimbra
Pólo II, Pinhal de Marrocos,
3030-290 Coimbra
PORTUGAL
E-mail:`{macpedro, edmundo, boavida}@dei.uc.pt`

Abstract

Inter-domain traffic engineering is a key issue when QoS-aware resource optimization is concerned. Mapping inter-domain traffic flows into existing service level agreements is, in general, a complex problem, for which some algorithms have recently been proposed in the literature. Nevertheless, these algorithms have not yet been the subject of extensive study. The purpose of this paper is to contribute to a better understanding of existing inter-domain optimization algorithms, by studying and comparing their performance in a variety of scenarios. The analysis presented in the paper showed that most algorithms fail in complex scenarios. To cope with this, several modifications to existing algorithms were proposed and tested. In addition, the analysis has shown that genetic based algorithms lead to better and lower cost solutions, at the expense of processing time.

1. Introduction

The main purpose of inter-domain resource optimization is to map incoming inter-domain traffic flows into inter-domain network resources, satisfying quality of service (QoS) requirements, while aiming at optimizing the use of network resources across autonomous systems (AS) boundaries. Network resources usage is, in any case, conditioned by the existing Service Level Specifications (SLSs) that, in turn, result from the Service Level Agreements (SLAs) established by each domain with its neighbors. For the purpose of this paper, the terms 'domain' and 'autonomous system' are synonyms.

In order to describe the inter-domain relationships of an autonomous system, one can build a simple model as shown in Figure 1. An autonomous system is interconnected to other autonomous systems by means of its ingress and egress interfaces. The service offerings between autonomous systems as well as their mutual responsibilities are described by means of Service Level Agreements. In general, each SLA defines a set of contractual, administrative and technical requirements. The latter are called Service Level Specifications.

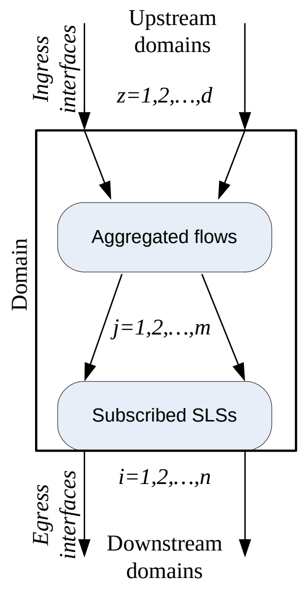

Figure 1. Inter-domain relationship model

An SLS comprises several items or clauses, including identification, application scope, flow identification, traffic conformance, excess treatment, and performance guarantees. The SLS identification allows for the unique identification of the SLS in the context of the SLA. The application scope identifies the region over which the QoS policy is enforced, normally by specifying their boundary ingress and egress points. The flow identification defines the kind of data stream, the source and destination information, as well as the respective QoS classes. On the other hand, the traffic conformance clause describes the characteristics that the traffic injected by the upstream domain should comply with in order to get the specified QoS guarantees. The excess treatment clause describes how to the excess traffic, i.e. out-of-profile traffic, should be processed. Lastly, the performance guarantees clause describes the packet transfer performance metrics that the domain should offer to its partners.

In the context of the present work, an SLS is characterised by a destination prefix, an inter-domain QoS class, *i-QC* as proposed in [20], the corresponding maximum bandwidth requirements, *bw*, and an egress interface. That is, an SLS entry for domain A is:

SLS entry = [(destination prefix) ∉ A, i-QC, bw, egress interface]

On the other hand, a domain receives from upstream domains a collection of *d* data flows towards other domains. Depending on the domain policy and on their common characteristics, such as destination and QoS class, these flows may be aggregated into *m* inter-domain traffic flows. The flows' common characterization includes the inter-domain class mapping and the destination prefix. That is, an aggregated flow entry for a domain A is:

Aggregate flow entry = [ingress interface, (destination prefix) ∉ A, i-QC, rbw]

where *rbw* is the bandwidth requirement of the aggregated flow. Aggregated flows must, then, be mapped into the existing SLSs. The appropriate selection of the SLSs for the inter-domain traffic flows benefits the domain by improving the network resources utilization. This task is executed today in a trial-and-error fashion.

The optimization of inter-domain network resources falls into the Generalized Assignment Problem (GAP) category, where the objective is to find a minimum cost assignment of $m > 0$ jobs to $n > 0$ agents, subject to the agents' available capacity. In the specific problem in hand, jobs are aggregated traffic flows and agents are the established SLSs.

Formally, the problem can be stated as follows. Let $I = \{1,2,...,n\}$ be the set of SLSs and $J = \{1,2,...,m\}$ the set of aggregated traffic flows. For each SLS *i* there is a given resource capacity, expressed in terms of bandwidth, $b_i > 0$. For each $i \in I$ and each $j \in J$ there is a given set of costs, $c_{i,j} > 0$, and resource requirements, $r_{i,j} > 0$, for assigning an aggregated traffic flow *j* to an SLS i. Additionally, $x_{i,j}$ is a variable that is set to 1 if the traffic flow *j* is assigned to SLS *i* and 0 otherwise. The mathematical formulation is as follows:

$$c(x) = \sum_{i=1}^{n}\sum_{j=1}^{m} c_{i,j} \cdot x_{i,j} \qquad (1)$$

subject to

$$\sum_{j=1}^{m} r_{i,j} \cdot x_{i,j} \le b_i, \forall i \in I, \qquad (2)$$

$$\sum_{i=1}^{n} x_{i,j} = 1, \forall j \in J, \qquad (3)$$

$$x_{i,j} \in \{0,1\}, \forall i \in I, \forall j \in J. \qquad (4)$$

The optimization goal is to minimize the cost (1), where the capacity constraint (2) ensures that the total resource requirements of the traffic flows assigned to each SLS do not exceed the available capacity. The assignment constraint (3) guarantees that each traffic flow is assigned to exactly one SLS.

The objective of the work presented in this paper is to evaluate and compare the more significant inter-domain traffic optimization algorithms. This will lead to a better understanding of the general problem of inter-domain traffic management and to an identification of the main obstacles to efficient and effective inter-domain QoS provision. In order to reach the stated objective, a set of test scenarios was generated. These scenarios take into account relevant work by other researchers on the characterisation of the Internet hierarchy [1], and were used to compare the optimization algorithms proposed in the MESCAL European project [2]. At a subsequent stage some improvement on the algorithms proposed in [2] was introduced.

Section 2 of this paper presents a description of each of the tested traffic engineering algorithms. In section 3 the evaluation framework is detailed along with some implementation strategies. Section 4 presents and discusses the obtained comparison results. Section 5 summarizes the work and presents future research directions.

2. Traffic engineering algorithms

Several studies on intra-domain resource optimization, such as [4][6][7][8], can be found in the literature. In the case of inter-domain, references [2][5][9] and [14] constitute the framework for most of the current proposals. Nevertheless, to the best of the authors' knowledge, there is a clear lack of a comprehensive study for the evaluation and comparison of the existing proposals for inter-domain resource optimization.

GAP is an NP-Complete problem which is not only unsolvable [18], but also very difficult to solve approximately [17][19]; thence the only option is relying on heuristics.

The use of greedy heuristic algorithms for selecting egress routers for inter-domain traffic with bandwidth guarantees, optimising the total bandwidth consumption in the network, is proposed in [5]. In this work the authors evaluate the use of three heuristic algorithms: greedy-cost, greedy-penalty, and greedy-random.

The optimization of inter-domain resource utilisation is being studied in MESCAL project [9]. In the context of this project Morand et al. [2] proposed six different algorithms for inter-domain resource optimization: random assignment, brute force, a genetic algorithm, and the three heuristic algorithms already addressed in [5] (greedy-cost, greedy-penalty, and greedy-random). Morand et al. also propose modifications to these algorithms in order to cope with resource exhaustion situations. In this context, one of the possible solutions is the generation of new SLSs.

Following the MESCAL work, our proposal is to evaluate algorithms for inter-domain traffic optimization, namely random assignment, brute force, greedy-cost, greedy-random, greedy-penalty and the combination of a genetic-brute force algorithm.

Genetic algorithms have already been extensively used to solve network optimization problems [4][6][7][8][17]. These algorithms, belonging to the class of evolution strategies used in optimization, resemble the process of biological evolution, where each individual is described by its genetic code, called a chromosome. On the other hand each chromosome is composed of individual genes. In the problem in hand, a gene is the assignment of a single aggregate traffic flow to an SLS, and an individual (i.e., a chromosome) is a potential solution.

The combination of genetic and non genetic algorithms has already been studied by several authors [15][16]. In these proposals non genetic algorithms are used to generate the initial set of feasible solutions. In our proposal we use the brute-force algorithm to generate the initial solution.

2.1. Optimization algorithms

This section described the algorithms to be evaluated in this paper.

2.1.1. Random assignment

This algorithm is as follows: each aggregated traffic flow, of *m* flows, is randomly assigned to an SLS, out of *n* SLSs that meet the required inter-domain class, destination prefix, and which have enough spare bandwidth to support the flow's data rate. If any of the flows cannot be assigned, the assignment process is restarted and a further attempt is made, up to a maximum of *M* attempts. The algorithm has a computational complexity of $O(Mmn)$.

2.1.2. Brute force

To get a first feasible solution, without taking the cost into account, a simplified version of a brute force algorithm was built. The algorithm determines all possible combinations of mappings between flows and SLSs, till a maximum number of combinations *c*. Then it tests each combination in order to check if the bandwidth requirement is met. Finally, when a first feasible solution is found, the algorithm stops.

This algorithm has a computational complexity of $O(2^c cmn)$.

2.1.3. Greedy-cost algorithm

This algorithm is based on the proposal presented in [2]. It starts by sorting the m aggregate traffic flows in descending order, based on their bandwidth requirements, and selecting the flows one by one, in that order. Then it performs the following steps:

Step 1 – A pre-selection of SLSs is made, where each SLS is evaluated individually in order to determine its ability to support the traffic flow. This pre-selection is based on information such as destination address prefix, QoS class, and available bandwidth. The assignment is feasible if the SLS meets the first two parameters and it has enough bandwidth to accommodate the flow's data rate.

Step 2 – Among the set of feasible SLS assignments determined in the preceding step, the SLS with the minimum cost is selected. The cost of assigning the aggregated traffic to an SLS is determined by the objective function defined in equation (5), below.

Step 3 – The next traffic flow is selected and steps 1 and 2 are repeated, until all traffic flows have been considered.

This algorithm has a computational complexity of $O(mn)$, where n is the number of SLSs.

2.1.4. Greedy-random algorithm

This algorithm is identical to the greedy-cost algorithm, with the exception of the step 2. In this step the selection among the set of feasible SLSs is done at random. If any of the flows cannot be assigned, the assignment process is restarted and a further attempt is made, up to a maximum of *M* attempts, as in the case of the random assignment algorithm. This algorithm also has a computational complexity of $O(Mmn)$.

2.1.5. Greedy-penalty algorithm

This algorithm is based on the proposal presented in [2]. It performs the following steps:

Step 1 – For each traffic flow calculate the desirability of assigning it to each SLS that meets the required destination address prefix, QoS class and available bandwidth. The desirability is the inverse of the cost (see equation (5), below) of assigning the flow to a specific SLS.

Step 2 – Evaluate the penalty for each traffic flow assignment, which is the difference between the desirability of the traffic flow's best and second best selections. If there is only one feasible SLS to accommodate the traffic flow, assign the flow to this SLS.

Step 3 – From all currently unassigned traffic flows, select the one yielding the largest penalty and assign it to the SLS that corresponds to the highest desirability. If multiple traffic flows have the same largest penalty, they are placed in order of decreasing bandwidth requirement.

Steps 1 to 3 are repeated until all traffic flows have been considered. This algorithm has a computational complexity of $O(mn)$, as in the case of the greedy-cost algorithm.

2.1.6. Genetic-brute force algorithm

The genetic algorithm is based on the proposal presented in [2] with some modifications. As shown in Figure 2, the algorithm starts by generating an initial random population in which is inserted a brute-force generated feasible solution. Then it follows a predefined number of generations (*G*). On each generation, the fitness of every chromosome is evaluated by (1) using the objective function given in equation (5). Then chromosomes are classified and divided into three sections: the best, the medium, and poor ones. The best chromosomes are saved and following an elitist procedure they are passed unchanged to the next generation. On the other hand the poorest are discarded. Next a process of crossover and mutation are applied to the best and the medium chromosomes to generate new chromosomes for the next generation. This process results in a new population of *N* chromosomes.

In the crossover process, two individuals are randomly selected, one from the best section of the population, and one from the medium section. The genes are randomly selected from the fitter parent with a probability of *pc*, plus the remaining genes coming from the other parent chromosome. At last each offspring chromosome is submitted to a mutation process where its genes are randomly changed with a mutation probability *pm*.

```
Generate the initial N chromosomes
population;
For G generations
   Evaluate fitness of each chromosome;
   Classify chromosomes as: best, medium,
   poor;
   Save the generation's best chromosome;
   Pass best chromosomes to next
   generation;
   Discard poor chromosomes;
   Crossover best and medium chromosomes;
   Mutate the offspring's;
End for
Choose best chromosome;
```

Figure 2. Genetic-brute force algorithm

The effectiveness and convergence rate of the genetic algorithm depends on the values of *N*, *pc* and *pm*. In the present study, the chosen values were the ones suggested by a previous research study by Lin et al. [3]. These are 150 <= *N* <= 300, 0.5 <= *pc* <= 0.8, and 0.001 <= *pm* <= 0.1.

The computational complexity of the simple genetic algorithm is of $O(GNmn)$, where *G* is the number of generations and $O(2^c cGNmn)$ when the brute-force algorithm was included to generate an initial feasible solution.

2.2. Objective function

The following cost function, $c_{i,j}$, or objective function, was used in order to measure the egress interfaces bottleneck:

$$c_{i,j} = \frac{1}{\left(bw^i_{sls} - bw^j_{flow} + 0.1\right)^2} \qquad (5)$$

where bw^i_{sls} is the available bandwidth on egress interface *i* (the agreed SLS) for some QoS class and destination, and bw^j_{flow} is the bandwidth of the aggregate flow *j*. The value 0.1 was added to the denominator in order to limit the value of $c_{i,j}$ to 100. This cost function was used in all algorithms.

3. Evaluation framework

This section describes the framework used to evaluate the traffic engineering algorithms.

3.1. Test scenarios

In order to evaluate each algorithm, several scenarios were built, each representing a typical autonomous system.

The characterisation of autonomous systems has been studied in several pieces of work in the recent past [1][12][13]. For the present study, the characterisation presented in [1] was used as a basis, as this represents the most recent work.

According to Subramanian et al. [1], ASs can be classified as costumer ASs, small regional ISPs, outer core, transit core, and dense core. The existing numbers of ASs, for each of these categories, are presented in Table 1. Still according to the same authors, there are certain numbers of ingress and egress interfaces to ASs of the same or different class. These numbers are reproduced in Table 2 below, where the columns represent ingress interfaces and the rows represent egress interfaces. For instance, there are 312 ingress interfaces between all dense core ASs (i.e., level 0 ASs), 183 ingress interfaces between dense core and transit core ASs, and 29 ingress interfaces between dense core and outer core ASs.

Using the values of Tables 1 and 2 as a base, four different evaluation scenarios were constructed, each one representing a typical non stub autonomous system of type ‘small Regional ISPs’, ‘outer core’, ‘transit core’, and ‘dense core’. ASs of type ‘customer’ as stub ASs were not considered in the tests.

Level	Layer	# of ASs
0	Dense Core	20
1	Transit core	129
2	Outer core	897
3	Small regional ISPs	971
4	Customers	8898

Table 1. Hierarchical distribution of ASs [1]

Each of these four scenarios (i.e., typical AS) is characterised by a given number of ingress interfaces, egress interfaces, supported QoS classes, maximum number of destination prefixes, aggregate bandwidth and number of aggregate flows, as presented in Table 3.

In this table, the number of destination prefixes corresponds to the total number of ASs, which can be found by adding the number of ASs given in Table 1.

Level	0	1	2	3	4
0	312	626	1091	958	6732
1	183	850	1413	665	3373
2	29	145	1600	543	3752
3	0	0	0	212	2409

Table 2. Interconnectivity between levels [1]

The number ingress interfaces in Table 3 was found by dividing the total number of ingress interfaces presented in Table 2 by the number of corresponding ASs presented in Table 1 (for instance, in the case of dense core ingress interfaces, 26 = (312 + 183 + 29) / 20). The number of egress interfaces was given applying the same process to egress interfaces of the table.

AS level	Dense core	Transit core	Outer core	Small Regional ISP
Test scenario	#0	#1	#2	#3
Ingress Interfaces	26	13	5	2
Egress Interfaces	486	50	7	3
QoS classes	3	3	3	3
Destination prefixes	10915	10915	10915	10915
Bandwidth (aggregated)	0..100	0..100	0..100	0..100
Aggregate flows	454997	227576	65168	42869

Table 3 – Test scenarios characterization

A maximum of 3 inter-domain QoS classes was selected. The maximum bandwidth per aggregate flow is expressed by a number between 0 and 100 (like a percentage).

The domain characteristics defined above were used by a flow generator algorithm in order to determine the number of aggregate flows for each of the scenarios. This algorithm consists of the following three steps:

```
Step 1: Create a set of ingress traffic
        flows;
Step 2: Aggregate the flows with the same
        class requirements and the same
        destination;
Step 3: Randomly and uniformly distribute
        the m aggregate flows by n egress
        interfaces;
```

In Step 1, the creation of the ingress traffic flows is done by randomly distributing, using a uniform distribution, a maximum of *F* flows per each of the *q* inter-domain classes and per *m* ingress interfaces, for *F* destinations prefixes, where *F* is the total number of autonomous systems previously defined.

Step 2 generates an aggregate traffic flow matrix, TM:

$$TM = \begin{bmatrix} j_1 & q_1 & k_1 & b_1 \\ . & . & . & . \\ . & . & . & . \\ j_m & q_m & k_m & b_m \end{bmatrix}$$

where,
j – Ingress interface
q – Inter-domain QoS class
k – Destination prefix
b – Flow's bandwidth
m – Number of flows

Step 3 leads to an SLS matrix, SM:

$$SM = \begin{bmatrix} i_1 & q_1 & k_1 & b_1 \\ . & . & . & . \\ . & . & . & . \\ i_n & q_n & k_n & b_n \end{bmatrix}$$

where,
i – Egress interface
q – Inter-domain QoS class
k – Destination prefix
b – SLS's bandwidth
n – Number of SLSs

For each generated scenario this algorithm guarantees that there exists at least one global assignment solution, that is, that it is possible to assign all traffic flows to SLSs.

3.2. Implementation issues

As seen above, during the generation process flows at a given ingress interface are aggregated by QoS class and destination prefix. Nevertheless, in the assignment of flow aggregates to SLSs, the ingress interface is irrelevant, the relevant parameters being the class, the destination and the required bandwidth. This leads to the creation of a higher grouping of flows, called aggregate block. An aggregate block is a group of aggregate flows with the same class and destination, irrespective of the ingress interface.

Similarly, SLSs can be grouped by class and destination, irrespective of egress interface. This is called an SLS block.

This organisation is depicted in Figure 3, where for each block of aggregated flows it is assumed that it exists one block of SLSs.

On the other hand, it is also assumed that there exists one, and only one, aggregate flow per egress interface belonging to some class and destination prefix. As a consequence, the maximum number of flows per block is limited by the number of egress interfaces.

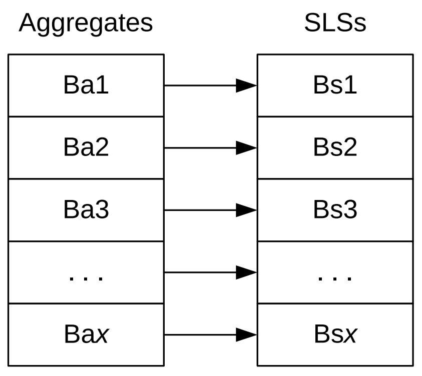

Figure 3. Aggregate blocks and SLS blocks

In the scenarios under study the number of ingress interfaces is lower than the number of egress interfaces and, for the sake of simplicity, only one flow per class and destination per each ingress interface was set up. As a result, the maximum block size in the test framework is limited by the number of ingress interfaces. In test scenario #0, with 454997 aggregated flows, the biggest block had 22 flows and the smallest had only 1 flow. The flow generator algorithm returned for each of the 4 test scenarios blocks ranging from 1 flow/1 SLS to 22 flows/22 SLSs. The number of blocks of scenarios #0, #1, #2 is depicted in figures 4 to 6 respectively. For scenario #3 we have only blocks of sizes 1 and 2.

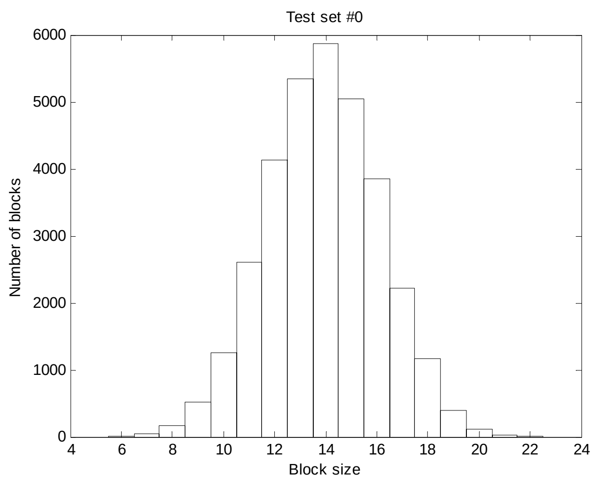

Figure 4. Number of blocks as a function of block size for scenario #0

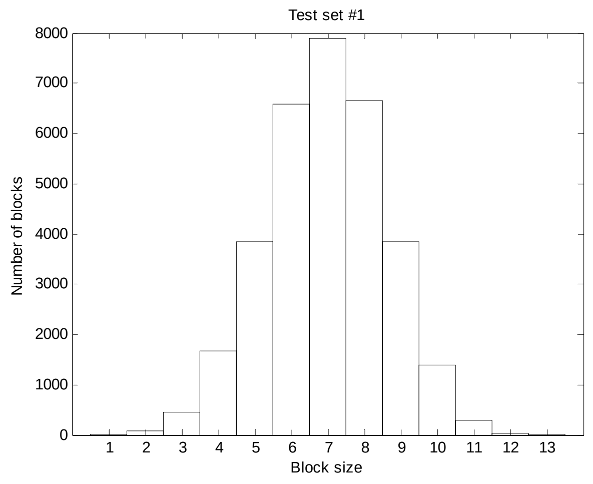

Figure 5. Number of blocks as a function of block size for scenario #1

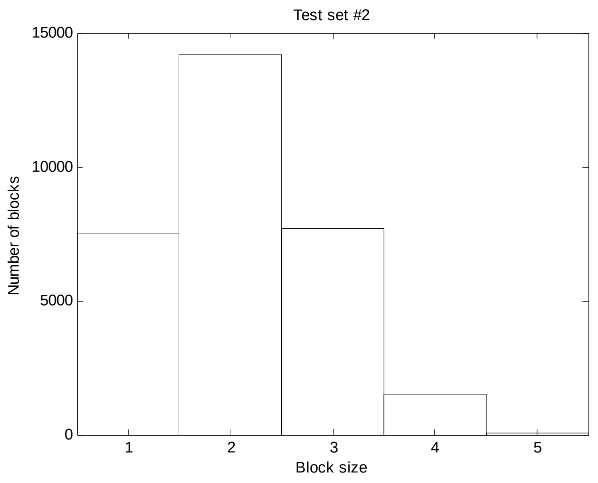

Figure 6. Number of blocks as a function of block size for scenario #2

The algorithms under study were coded using the MATLAB language, and the tests were performed in a machine with a Pentium 4 processor at 1.7 GHz, 1GBytes of RAM, with the MS Windows XP operating system.

3.3. Algorithms comparison

The comparison of the various algorithms under study was carried out using the following parameters:

1. total cost of the solution, given by equation (1);
2. the cost for each block of aggregated flows, given by equation (1) to the aggregate flows in the block;
3. total processing time spent by the test machine in order to run the algorithms for each test scenario;
4. processing time spent by the test machine in order to run the algorithms for each block of aggregated flows.

A maximum of 1000 attempts were set up for the random assignment algorithm and for the greedy random algorithm. The brute force algorithm was limited to 10 levels of search combinations. The genetic-brute force (Genetic-BF) algorithm includes one feasible brute-force generated solution on its initial population and was configured with a maximum of 500 generations, with 150 individuals, crossover probability of 0.6 and mutation probability of 0.05.

4. Results

Each of the four scenarios presented in Table 3 was used for testing each of the assignment algorithms presented in Section B. The results of the tests are presented below.

Table 4 presents the total cost of the solution returned by each algorithm for each scenario. The letter F in the table means that the algorithm fail to return an assignment solution (meaning that, considering all blocks in the scenario, there exists at least one for which the algorithm failed).

Algorithms	Scenario 0 (x1e7)	Scenario 1 (x1e7)	Scenario 2 (x1e6)	Scenario 3 (x1e6)
Random	F	F	5,86	3,84
Greedy-cost	F	F	F	3,85
Greedy-random	F	F	5,88	3,85
Greedy-penalty	F	F	F	3,85
Genetic-BF	4,47	2,13	0,59	3,84
Brute force	4,49	2,14	5,88	3,85

Table 4. Total cost of the obtained assignment solutions

In this table, we verify that for scenario 3 the random and the genetic algorithms lead to the lowest cost, with a value of 3.84×10^6. For the scenario 2 the greedy-cost and greedy-penalty algorithms fail and the Genetic-BF algorithm returns the lowest cost solution. Lastly, for the remaining scenarios, 0 and 1, only the brute force and the Genetic-BF algorithms return an assignment solution. The Genetic-BF algorithm always leads to the lowest cost solutions.

Figure 7 provides a view on the cost gain of the Genetic-BF algorithm over the brute force algorithm, as a function of block size. The presented values are the cost difference between the value returned by the Genetic-BF algorithm and the value given by the brute force algorithm. These are average costs of more than 32000 blocks, with sizes ranging from 1 flow to 22 flows.

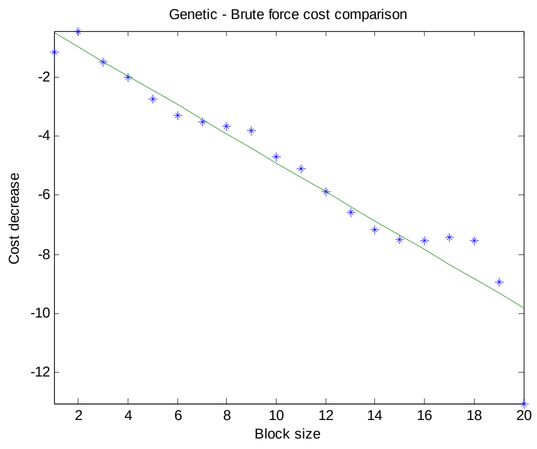

Figure 7. Genetic – Brute force cost gain

Table 5 presents the processing time spent by each algorithm, per each scenario. Once more, the letter F appears for the algorithms that could not return an assignment solution.

For scenario 3, the lowest processing time is obtained with the greedy-cost algorithm. The Genetic-BF algorithm leads to the highest processing time. In scenario 2, among the successful algorithms the Genetic-BF algorithm leads to the highest processing time and the brute force algorithm to the lowest. This characteristic persists in the remaining scenarios.

Algorithms	Scen. 0 (x1e5)	Scen. 1 (x1e4)	Scen. 2 (x1e4)	Scen. 3 (x1e2)
Random	F	F	0,35	21,30
Greedy-cost	F	F	F	0,41
Greedy-random	F	F	0,36	20,90
Greedy-penalty	F	F	F	0,54
Genetic-BF	1,06	1,15	0,59	44,80
Brute force	0,88	0,12	0,01	0,72

Table 5. Total processing time in seconds

Figure 8 presents a comparison between the average processing times of the brute force algorithm and the Genetic-BF algorithm, as a function of block size. These values were given for the same blocks of the previous figure. The curve of Genetic-BF (G-BF) algorithm is the sum of the 'Brute force' and simple genetic algorithm (GA) curves.

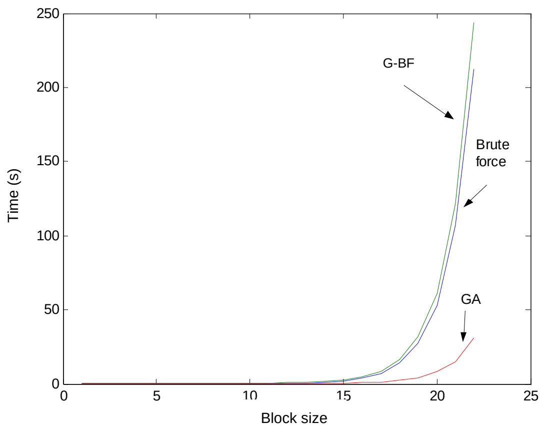

Figure 8. Processing times

Summarising the results, one can highlight the following:

The algorithms proposed in [2] fail in scenarios with larger size blocks, such as scenarios #0 and #1 (that is, scenarios with more than 5 flows per block).

The brute-force and the genetic-brute force algorithms lead to assignment solutions in all scenarios. As depicted in Figure 7, the Genetic-BF algorithm always leads to lower cost solutions. Additionally, the same figure clearly shows a tendency to a decreasing cost of the assignment solutions cost with the increase of the size of the processed blocks.

On the other hand, the Genetic-BF algorithm has the highest processing times. Nevertheless, as showed in Figure 8, most of the time is spent by the brute force algorithm to find a first assignment solution on which the genetic algorithm can improve.

5. Conclusion

Inter-domain QoS-aware resource optimization is one of the main challenges of current traffic engineering. Starting with a set of algorithms for inter-domain resource assignment recently proposed in the literature, this paper presented extensive testing of these algorithms and proposed some modifications in order to improve their performance.

As a general conclusion, one can say that most of the existing resource optimization algorithms fail in complex scenarios. The proposed genetic-brute force algorithm can lead to better solutions than the brute-force algorithm alone, although at the cost of high processing time. Nevertheless the additional processing time of the simple genetic algorithm is only a fraction of the processing time of the simple brute-force algorithm.

The obtained results clearly show that there is still a long way to go in order to reach a situation where inter-domain resource optimization can easily be achieved.

Further work of the authors will address the main limitations of the algorithms under test identified in the presented work, which lead to extremely high processing time. This work will explore the use of other versions and combinations of genetic algorithms, like the hybrids proposed in [10][11], as well as the inclusion of other QoS parameters in the decision process as, for instance, delay and loss.

Acknowledgement

This work was partially carried out with the support of the E-NEXT Network of Excellence (Emerging Networking Experiments and Technologies, IST-FP6-506869)

References

[1] L. Subramanian, S. Agarwal, J. Rexford, and R. Katz. Characterizing the Internet hierarchy from multiple vantage points. *Proc. IEEE INFOCOM*, 2002.

[2] P. Morand, et al. D1.1: Specification of Business Models and a Functional Architecture for Inter-domain QoS Delivery. IST-2001-37961, unpublished, June 2003. [Online]. Available: www.mescal.org/

[3] X. Lin, Y. Kwok and V. Lau. A genetic algorithm based approach to route selection and capacity flow assignment. *Computer Communications*, Vol. 26, pp.961-974, 2003.

[4] M. Ericsson, M. Resende, and P. Pardalos. A genetic algorithm for the weight setting problem in OSPF routing. ATT Shannon Laboratory 180 Park Avenue Florham Park, NJ 07932, Tech. Rep., 2001. [Online]. Available: citeseer.ist.psu.edu/526656.html

[5] K. Ho, N. Wang, P. Trimintzios, G. Pavlou, M. Howarth. On Egress Router Selection for Inter-domain Traffic with Bandwidth Guarantees. *Proceedings of the IEEE Workshop in High Performance Switching and Routing (HPSR'2004),* Phoenix, Arizona, USA, IEEE, April 2004.

[6] L. Buriol, M. Resende, C. Ribeiro, and M. Thorup. A memetic algorithms for OSPF routing. In *Proc. 6th INFORMS Telecom*, pp. 187--188, 2002.

[7] L. Buriol, , M. Resende, C Ribeiro, and M. Thorup. A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. unpublished, 2003. [Online]. Available: www.optimization-online.org/DB_FILE/2003/06/674.pdf

[8] A. Riedl. A hybrid genetic algorithm for routing optimization in IP networks utilizing bandwidth and delay metrics. *IEEE Workshop on IP Operations and Management* (IPOM), Dallas, October 2002.

[9] MESCAL project [website]: www.mescal.org.

[10] P. Chu and J. Beasley. A genetic algorithm for the generalised assignment problem. *Computers & Operations Research*, 24:17–23, 1997.

[11] G. Raidl and H. Feltl. An improved hybrid genetic algorithm for the generalized assignment problem. In H. M. Haddadd et al., editors, *Proceedings of the 2003 ACM Symposium on Applied Computing,* pages 990-995. ACM Press, 2004.

[12] L. Gao. On inferring autonomous system relationships in the Internet. *IEEE/ACM Trans. Networking,* vol. 9, no. 6, December 2001.

[13] Z. Ge, D. Figueiredo, S. Jaiwal, L.Gao. On the hierarchical structure of the logical Internet graph. In *Proceedings SPIE ITCOM,* August 2001

[14] T. Bressoud, R. Rastogi, and M. Smith. Optimal Configuration for BGP Route Selection. In *Proceedings of IEEE INFOCOM' 2003,* San Francisco, March/April 2003.

[15] R. Ahuja, J. Orlin, and A. Tiwari. A greedy genetic algorithm for the quadratic assignment problem. *Computers and Operations Research*, vol. 27, pag. 917–934, 2000.

[16] S. Areibi, and A. Vannelli. Efficient hybrid search techniques for circuit partitioning. In *proc. IEEE 4th World Multiconference on Circuits, Systems*, Communications & Computers, 2000.

[17] M. Pioro and D. Medhi. *Routing, Flow, and Capacity Design in Communication and Computer Networks*. Morgan Kaufmann Series in Networking, 2004.

[18] T.Cormen, C. Leiserson, R. Rivest and C. Stein. *Introduction to Algorithms*, Second Edition, MIT Press, 2001.

[19] D. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, vol. 62, pp. 461–474, 1993.

[20] P. Levis, M. Boucadair, P. Morand, P. Trimintzios. The Meta-QoS-Class concept: a step towards global QoS inter-domain services. *Proc. of IEEE International Conference on Software, Telecommunications and Computer Networks* (SoftCOM 2004), 2004.