A multi-deme parallelization of a memetic algorithm for the weight setting problem in OSPF and DEFT

Patrick Heckeler[1], Marcus Ritt[2], Luciana S. Buriol[2], Mauricio G.C. Resende[3], Wolfgang Rosenstiel[1]

[1] Wilhelm-Schickard-Institut für Informatik
Eberhard Karls Universität
Tübingen, Germany
heckeler@informatik.uni-tuebingen.de,rosenstiel@informatik.uni-tuebingen.de

[2] Instituto de Informática
Universidade Federal do Rio Grande do Sul
Porto Alegre, Brazil
mrpritt@inf.ufrgs.br, buriol@inf.ufrgs.br

[3] Algorithms and Optimization Research Department
AT&T Labs Research
Florham Park, United States
mgcr@research.att.com

Abstract

The Internet is divided into Autonomous Systems, which control their intra-domain traffic by using interior gateway protocols. The most common protocol used today is Open Shortest Path First (OSPF). OSPF routes traffic on shortest paths defined by integer link weights. The weight setting problem is to find weights that optimize the resulting traffic, for example to minimize network congestion. A recently proposed protocol called Distributed Exponentially-weighted Flow Splitting (DEFT) sends flow on non-shortest paths, with an exponential penalty for longer paths. Since these problems are hard to solve exactly, several heuristics have been proposed. We propose a parallel, multi-deme version of a memetic algorithm to solve the weight setting problem in DEFT. It consists of a shared memory parallelization of the (single deme) memetic algorithm, as well as instances of the memetic algorithm running in parallel, and migrating solutions among populations according to the island model. Computational results show a reduction of execution time, and an improvement of solution quality compared to the original memetic algorithm.

1 Introduction

The Internet is divided into Autonomous Systems (ASs). Each AS controls its interior routing by an interior gateway protocol. Common interior gateway protocols, for instance Open Shortest Path First (OSPF), allow the operator to define the routes by setting integer weights on the network links. For a given protocol, the problem of finding weights which optimize some objective function, such as total network congestion, link utilization, or latency, is called the *weight setting problem*. In this article we focus on the current standard OSPF and a recently proposed protocol called Distributed Exponentially-weighted Flow Splitting (DEFT) [21].

Fortz and Thorup showed that the weight setting problem for OSPF is NP-hard and proposed a heuristic solution using tabu search [7, 8]. Several authors have proposed further heuristics solutions, including genetic algorithms [6], memetic algorithms [2], and simulated annealing [13]. Some of the best results for OSPF have been obtained by Tabu search [7] and a memetic algorithm [2].

For DEFT, Xu et al. [21] designed a heuristic two-stage iterative method, based on non-linear, non-smooth optimization, considering real weights. It is quite difficult to parallelize this method, since the solution technique is a modified primal-dual interior point filter line search [19]. Recently, Reis et al. proposed a memetic algorithm [15] for DEFT considering integer weights. In the comparison results of that paper, the authors show that, using identical available resources, int-DEFT produces less network congestion than OSPF routing does. However, int-DEFT produces solutions with longer path lengths, larger percentage of intermediate nodes, and larger number of paths. These are all undesirable characteristics

since a link failure will result in a larger expected number of affected O-D demand pairs. In a further proposal, Xu et al. [20] proposed a new link-state protocol called PEFT, which also considers real weights and splits traffic over multiple paths with an exponential penalty on longer paths, as DEFT does.

In this article, we consider the algorithm proposed in [15] and study to what extent multiple populations and parallelization can improve solution quality and reduce execution time compared to the sequential approach.

A genetic algorithm (GA), as first proposed by Holland [11], evolves a single population. A variation, which makes the model more realistic, is a multi-deme GA. Here, we let evolve multiple populations (demes) almost independently, and migrate individuals between demes to provide a weak interaction. From an evolutionary point of view, the model corresponds to punctuated equilibria or almost separately evolving "islands". For an overview of multi-deme approaches we refer the reader to [1]. The topology, frequency and rate of the migration has a strong influence on the performance of this approach [4].

There are two widely-used approaches to parallel genetic algorithms [1, 3]. In the single-deme approach, we can maintain a single global population, and parallelize central operations, such as fitness evaluation and crossover with local search, to speedup the execution. In genetic algorithms these are usually light-weight operations, which are executed several times. This leads to a fine-grained parallelism, well-suited for a multi-threaded implementation on shared memory machines. For a speedup on distributed memory machines, the parallelism has to be sufficiently coarse. This is usually achieved by a Master-Slave scheme, where a master repeatedly distributes parts of the population to all machines, which execute some operations in parallel, and the master afterwards collects the results.

The multi-deme variant of a genetic algorithm already has a coarse-grained structure with several populations, and limited interactions between them. Therefore, it has less communication demands and is better suited for a parallelization, especially on distributed memory machines (some authors call this a "distributed" genetic algorithm).

In this paper we study the single-deme and multi-deme variants of a memetic algorithm for the weight setting problem, both in a sequential and a parallel implementation. The execution has been parallelized on two levels. A shared memory parallelization speeds up the evolution of a single deme, and a distributed memory parallelization allows multiple demes to evolve in parallel.

The remainder of this paper is organized as follows. In Section 2 we introduce the protocols OSPF and DEFT. Next, in Section 3, we briefly present the MA proposed for solving the weight setting problem in OSPF and DEFT. In Section 4, we detail the parallelization of these two algorithms. The computational results of the parallel, multi-deme version are summarized in Section 5. Finally, Section 6 presents some conclusions, as well as possible future investigations.

2 The OSPF and DEFT protocols

Let $G = (V, E)$ be a directed graph with link capacities $c_{u,v}$, and D a demand matrix where D_{ij} denotes the traffic demand from source node i to destination node j, for $1 \leq i, j \leq |V|$. Let $T = \{v \mid D_{uv} > 0\}$ be the subset of nodes that are the destination of at least one demand pair.

The multi-commodity routing problem is to find flows $f_{u,v}$ which satisfy all demands and minimize the total link utilization

$$\text{minimize} \qquad \sum_{(u,v)\in E} \Phi(f_{u,v}, c_{u,v}) \tag{1}$$

where Φ is a link cost function. A typical choice for Φ is the piece-wise linear function shown in Figure 1 [7, 8].

Let $f^t_{u,v}$ be the flow on link (u, v) destined to node t. Then any resulting flow must respect the constraints of flow conservation at intermediate nodes $v \neq t$

$$\sum_{(u,v)\in E} f^t_{u,v} - \sum_{(v,w)\in E} f^t_{v,w} = D_{v,t} \tag{2}$$

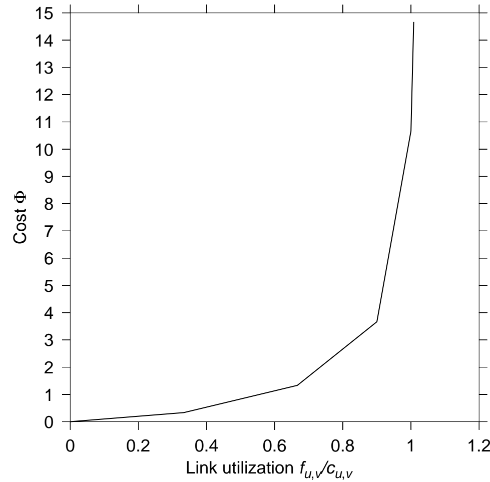

Figure 1: Link cost Φ depending on the link utilization for $c_{u,v} = 1$.

and the individual flow aggregation

$$f_{u,v} = \sum_{t \in T} f^t_{u,v}. \tag{3}$$

Since the objective function and all constraints are linear, we can find an optimal solution by solving the linear program OPT given by Eqs. (1), (2), and (3) together with the trivial constraints

$$0 \leq f^t_{u,v}, 0 \leq f_{u,v}. \tag{4}$$

The solution of OPT is called fractional multi-commodity flow routing. This kind of routing is not employed in practice, since it is difficult to implement and can lead to long paths and small link loads. Since OPT has no routing constraints, its solution serves as a lower bound for practical routing protocols.

In OSPF the flow is determined using integer weights $w_{u,v} \in [0, 2^{16} - 1]$ on each link. The routers exchange information about the links, including their weights. Each router uses these weights to compute the shortest paths to all destinations. It then distributes outgoing traffic destined to a node t equally among all outgoing links on shortest paths having t as destination.

DEFT relaxes this constraint. It allows real weights $w_{u,v} \in \mathbb{R}$ and distributes the flow amongst all outgoing links whose next node is closer to the destination. Links which are not part of a shortest path receive a flow which decreases with exponential penalties for longer path lengths. Formally, let d^t_i be the distance from node i to destination t, and let $h^t_{u,v} = d^t_v + w_{u,v} - d^t_u$ be the distance gap of using the link (u, v) compared to the shortest path. Then, the non-normalized traffic fraction Γ for link (u, v), directed to t, is calculated as

$$\Gamma(h^t_{u,v}) = \begin{cases} e^{-h^t_{u,v}} & \text{if } d^t_u > d^t_v \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

and the fraction of the total flow $\Gamma(h^t_{u,v}) / \sum_{v:(u,v)\in E} \Gamma(h^t_{u,v})$ is calculated for each outgoing link (u, v) of u. According to [21], in terms of total link cost and maximum utilization, there always exists a weight setting such that DEFT is better than OSPF.

Finding such weights, on the other hand, i.e. solving the weight setting problem optimally for these protocols is difficult. For example, finding the weights minimizing link utilization in OSPF is NP-hard [7].

3 A memetic algorithm for the weight setting problem

In this section we describe briefly the memetic algorithm previously proposed in the literature to solve the weight setting problem for OSPF and DEFT. More details can be found in [2, 15].

A memetic algorithm, or hybrid genetic algorithm, is a genetic algorithm augmented with a local search procedure to speedup the search by improving candidate solutions locally. In this context, a solution is called an *individual*, each element of the solution is a *gene*, a set of individuals is called a *population*, and each iteration of the algorithm is called a *generation*. It is a populational method in which, during each iteration, individuals are combined through a crossover procedure for generating new individuals that will form the next generation. The algorithm runs for a number of generations, aiming to improve the quality of solutions. Each solution is evaluated by an objective function that, in this problem, is to minimize the network congestion.

In our approach, each individual is represented by a vector of arc weights. The population is structured into three classes, according to their fitness, as first proposed by Ericsson et al. [6] in a genetic algorithm for OSPF routing. Class $\mathcal{A}$ contains the best 25% of the individuals, class $\mathcal{C}$ is composed by the 5% less profitable solutions, and the remaining population pertains to class $\mathcal{B}$. The solutions from class $\mathcal{A}$ pass directly to the next generation. The solutions from class $\mathcal{C}$ are replaced by new ones randomly generated. The remaining solutions are replaced by solutions generated by the crossover procedure between a random parent from class $\mathcal{A}$ and another from set $\mathcal{B} \cup \mathcal{C}$.

The crossover operator is a random key scheme that prioritizes (given 70% of chances) genes from parents in class $\mathcal{A}$. With a small probability of 1%, the child inherits a completely random allele at some given gene. We apply a local search on each solution generated by a crossover operator. This procedure is the computationally most expensive operation of the proposed MA. It examines the effect of increasing the weights of a subset of arcs. These candidate arcs are selected among those with the highest routing costs according to Φ function, and whose weight do not exceed the maximum allowed. To reduce the routing cost of a candidate arc, the local search attempts to increase its weight to induce a reduction on its load. If this leads to a reduction in the overall routing cost, the change is accepted, and the procedure is restarted. This procedure executes consecutive solution evaluations, that are expensive computational operations in this problem. To speedup this process, given a weight change, the shortest path graphs, as well as the flow allocation, are only updated, instead of recomputed from scratch. Updating, instead of recomputing from scratch, makes this procedure about 15 times faster.

The solution evaluation is the second most expensive operation of the proposed MA in terms of computational time. Given a set of integer weights, a shortest path graph G^t is computed, as well as the routing (flow allocation), for each destination node $t \in T$.

We apply the same memetic algorithm for the DEFT protocol, changing the evaluation procedure according to Equation 5. While OSPF splits the flow of each node u evenly among all outgoing links on shortest paths with destination t, DEFT splits the same load among *all* outgoing links (u, v) that approach t, i.e., $d_u^t > d_v^t$. Moreover, the load split is not equal among all links as it is in OSPF. As a consequence, changing the weight of an arc has a larger impact in DEFT, which increases considerably the computational effort in the dynamic flow calculation.

4 A parallel multi-deme variant of the memetic algorithm

In this section we describe a parallel, multi-deme variant of our memetic algorithm for the weight setting problem. The motivation for the multi-deme variant is to study whether it improves the solution quality, compared to the single-deme variant. The goal of the parallelization is to speedup execution or, equivalently, obtain better results in the same amount of time. Another minor goal was to keep the parallelization portable to a wide range of architectures to avoid the tedious task of adjusting the parallelization strategy for the hardware on which it is executed.

In a multi-deme GA, the migration operator is defined by a migration interval, a migration rate, a selection and replacement policy, and the migration topology [3, 18]. The migration interval defines

when migration happens, and is usually a fixed number of generations. The migration rate determines the number of migrants. Most commonly, this is a fixed percentage or number of individuals. According to the selection and replacement policy, the migrants are chosen from the source deme and integrated into the destination deme. Typical policies are random or fitness-based, e.g. migrating the best individuals and substituting the worst at the destination. A further option is to clone the migrants, or actually move them (termed immigration - resp. emigration by some authors). The possible destinations can be modeled by a directed graph, whose vertices represent the demes. An edge (u,v) of this graph connects deme u to v, if migration from u to v is possible. Frequently, migration topologies are low-dimensional grids (including cycles) and complete graphs [1]. From the possible destinations, we can choose one or more at random or employ some other scheme, for example a round-robin distribution.

Our multi-deme MA uses a (logical) unidirectional ring topology. We found little evidence on the influence of the migration topology on the solution quality and chose this topology, since the few studies available seem to indicate that the migration topology has less importance than other parameters [5]. The selection policy is to choose the fittest individuals. The replacement policy, the migration interval, and the migration rate have been determined experimentally (see next section).

To decrease execution time, we use a hybrid parallelization, which combines a shared memory and a distributed memory parallelization [12, 16]. The shared memory parallelization speeds up the evolution of a single deme, by executing the main steps of the genetic algorithm with multiple threads. This has been applied to solution evaluation, mutation, crossover and local search, the latter being the most time-consuming operation of the algorithm. All these operations can be done efficiently in a data-parallel fashion without synchronization between the threads. This part has been implemented with OpenMP, a high-level API for multi-threaded programming. [14]. The number of threads created by OpenMP can be easily tuned to the processor architecture to achieve an optimal performance.

The distributed memory parallelization applies to the multi-deme variant of the genetic algorithm. Each population is assigned to one computing node, which may consist of multiple processors, and uses the shared memory parallelization to evolve the local population. Migration uses message-passing between the nodes to send and receive the individuals. The parallelization has been implemented using Transmittable Parallel Objects (TPO++) [9, 10], an object-oriented communication library on top of the Message Passing Interface (MPI). TPO++ simplifies parallel programming on distributed machines in C++.

The overall communication structure is shown in Figure 2, for the example of four populations.

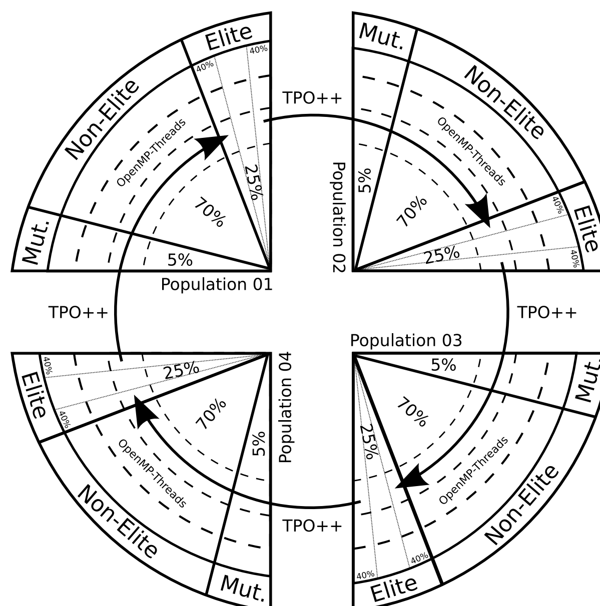

Figure 2: Structure of the hybrid parallelization with four populations. The dashed lines indicate the thread-level parallelism, the black arrows indicate the ring-shift communication based on TPO++.

5 Computational Results

We conducted a number of experiments to measure the effects of the hybrid parallelization on execution time and solution quality. From the instances available in the literature [8, 21], we have chosen four for our tests, as described in Table 1. Each instance defines all demand pairs in the network. Since the difficulty of solving the weight setting problem increases with the total demand, we have scaled the basic demands of each instance by factors of 6, 9 and 12.

Table 1: Instances used in the computational experiments.

Name	Instance	Nodes	Links	Capacities
`hier50a`	2-level hierarchy	50	148	200,1000
`hier50b`	2-level hierarchy	50	212	200,1000
`rand50`	Random topology	50	228	all 1000
`rand50a`	Random topology	50	245	all 1000

We conducted three sets of experiments for analyzing the results. The first set had the purpose of defining the parameters to be used in the following experiments. The other two experiments analyzed the speedup of the shared memory parallelization on a four-processor, and the speedup of the multi-deme parallel implementation. The experiments are reported in the next sections.

5.1 Parameter setting

In a preliminary experiment, we chose instance `hier50a` with a scale factor of 12 to determine the migration rate, the migration interval, and the replacement policy. In this experiment we have used 4 populations of 100 individuals and report the optimality gap after 50 generations (for this instance the algorithm achieves convergence with 50 generations). We tested migration intervals of 1, 2, 5 and 10 iterations, and migration rates of 1, 5 and 10 percent of the population. We also compared three different replacement policies: immigration, replacing worst elite solutions, immigration, replacing worst global solutions and emigration, replacing worst elite solutions. The result of these experiments are shown in Figures 3 to 5. The immigration replacement strategies turn out to perform better than emigration. When using immigration, replacing elite solutions gives better results than replacing globally worst ones. For both cases of immigration, we found that migrating 10% of individuals with a small number of iterations yielded the best results. We therefore chose to migrate 10% of the individuals every second iteration in the remaining experiments.

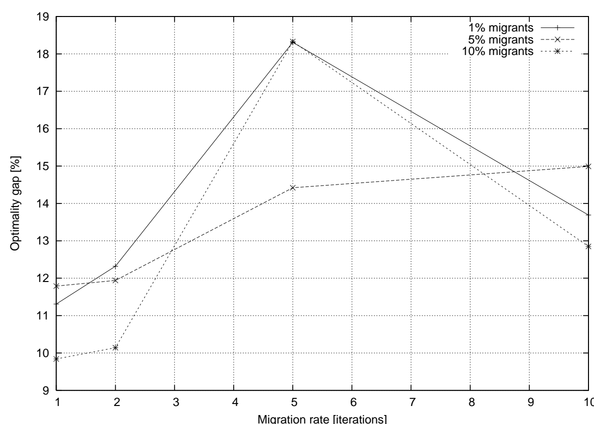

Figure 3: Optimality gap as a function of migration rate and migration interval and immigration replacing worst elite solutions.

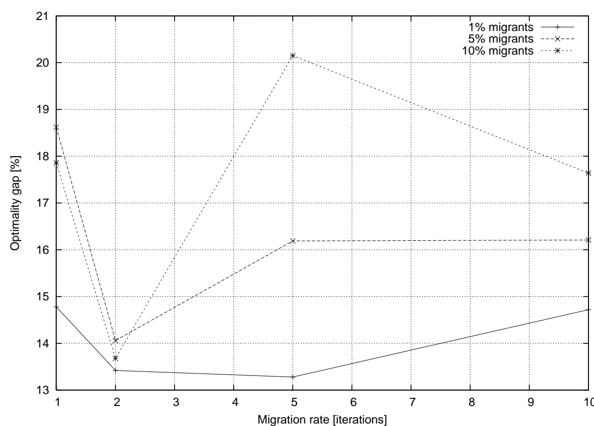

Figure 4: Optimality gap as a function of migration rate and migration interval and immigration replacing worst global solutions.

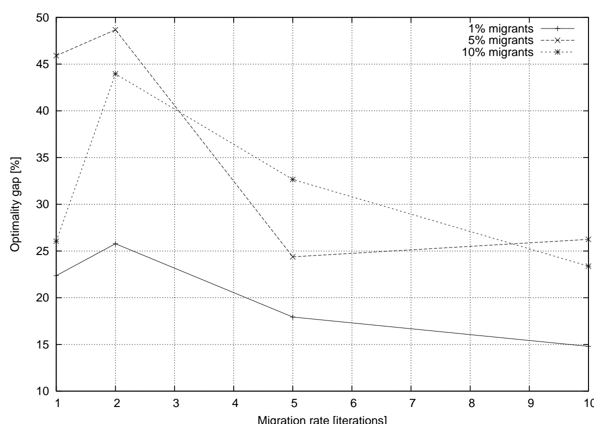

Figure 5: Optimality gap as a function of migration rate and migration interval and emigration replacing worst elite solutions.

5.2 Shared memory parallelization

In our first main experiment we measured the speedup of the shared memory parallelization on a four-processor AMD Opteron 275, with 4 GB main memory running at 2.2 GHz. We compared three different OpenMP loop scheduling strategies for the parallelization of the crossover and subsequent local search, which is the most time-consuming operation. Using static scheduling the loop iterations are distributed block-wise and evenly among all threads. The dynamic distributions repeatedly assign smaller blocks of 1 and 3 contiguous iterations to each idle thread, until all iterations have been completed.

Figure 6 shows the speedup for the three different scheduling strategies. All data points are the average over three executions. Clearly, all three scheduling strategies achieve good speedups of about three with six processors. The dynamic scheduling with smaller block-size reaches this speedup first, with only four processors, and shows less variation than the other strategies. Since the execution time of each local search can be different, a fine-grained load balancing seems to be the best strategy. Increasing the number of threads above six does not improve the results, which indicates that the speedup is not limited by some of the threads idling, but a not parallelized part of about 10% of the executed code.

5.3 Distributed-memory parallelization

Our last set of experiments is designed to quantify the utility of a multi-deme parallel GA compared to the single-deme - sequential GA, given the same amount of (wall clock) execution time. We measured the improvement of the solution quality as a function of the number of populations. Each population has

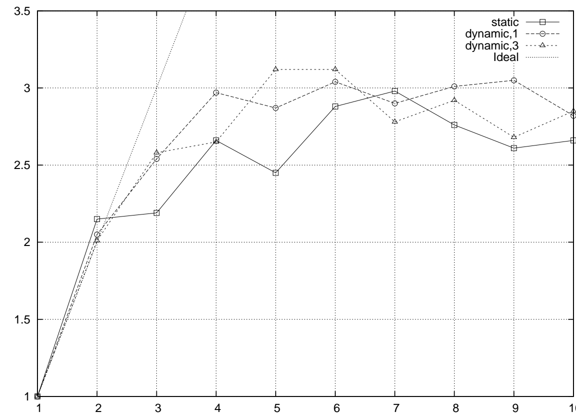

Figure 6: Speedup of the multi-threaded memetic algorithm on a four-processor SMP machine.

100 individuals, so the total population size is increasing with the number of populations.

The experiments have been conducted on the bwGRiD [17], a parallel computing grid consisting of a total of 498 nodes, each with two Intel Xeon E5540 processors running at 2.83 GHz, and equipped with 16 GB of main memory. The machine can communicate over Gigabit Ethernet and InfiniBand. In our tests we used the faster InfiniBand network.

We varied the number of populations from 1 to 256. To speedup the execution, each population evolves locally in parallel using 6 threads, but to quantify only the effect of the number of populations, we held the number of generations constant at 200.

Figures 7-8 show the best solution found over all populations as a function of the number populations, for instances `hier50a`, `hier50b`, `rand50` and `rand50a`, respectively. The solution values are given as the optimality gap in percent above the theoretical optimum, as determined by the linear program OPT described in Section 1.

In all four instances and three demand levels, the solution quality improves with an increasing number of populations. The improvements are more significant for instances with a higher total demand, and range from a small absolute improvement of 0.7% for instance `hier50a`/06 up to 100% for instance `rand50a`/12. Since the instances with a smaller total demand are simpler to optimize the absolute improvement with a larger number of populations is less articulate. In almost all instances, the solution quality did not improve significantly with more than 128 populations.

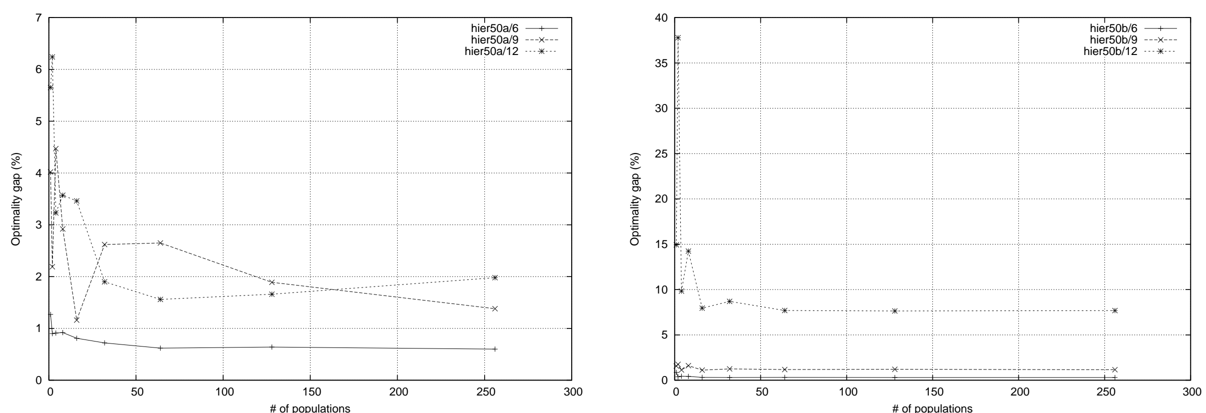

Figure 7: Solution quality as a function of the number of populations for instance `hier50a` (left) and `hier50b` (right).

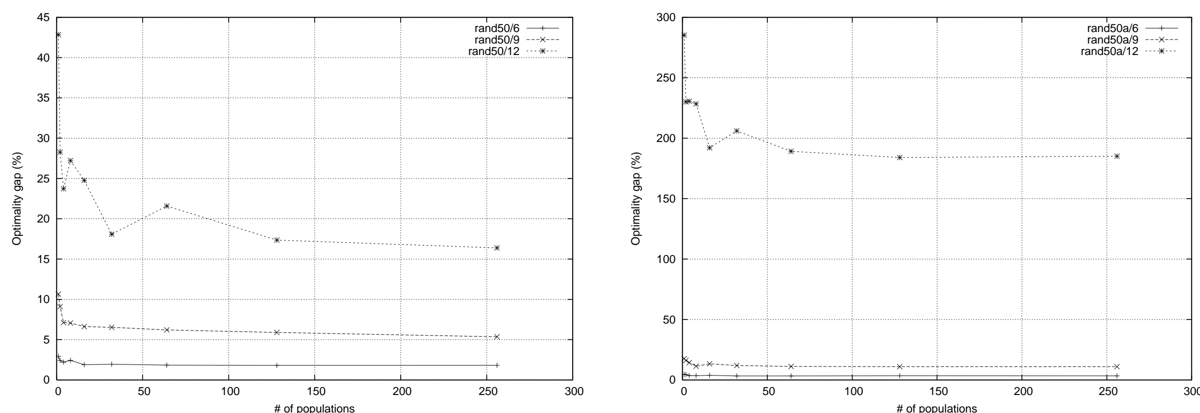

Figure 8: Solution quality as a function of the number of populations for instance `rand50` (left) and `rand50a` (right).

6 Conclusions and future work

We have studied two improvements of a memetic algorithm for the weight setting problem in OSPF and DEFT: a multi-deme variant of the algorithm as well as a shared-memory and distributed-memory parallelization.

We have shown that a good per-population speedup can be achieved with a shared memory parallelization using OpenMP. Parallelization of the principal loops of the memetic algorithm can speedup the evolution of a single population by a factor of up to 10.

The multi-deme, distributed memory parallelization based on MPI has improved the result quality for all tested networks. Our parallelization allows the memetic algorithm to increase the number of participating populations up to the number of available machines. The memetic algorithm could improve the solution quality consistently with the number of populations, although, in the tested instances, the improvement above 128 populations is only marginal. In a hybrid execution mode, combining shared-memory and distributed-memory we can optimally use a large class of parallel machines, reducing execution time and improving result quality.

As future work, we intend to test some other communication topologies with the aim of obtaining more benefit of the exchanged solutions among populations. We expect not just to increase solution quality, but also to reduce the time that the algorithm needs to converge. Furthermore, we intend to perform more detailed parameter studies to examine the behavior of the algorithm with more generations, different population sizes, and more complex network graphs.

Acknowledgement

The second and third authors thank the Brazilian National Council for Scientific and Technological Development (CNPq) for the financial support.

References

[1] Enrique Alba and Marco Tomassini. Parallelism and evolutionary algorithms. *IEEE Trans. on Evolutionary Computation*, 6(5):443–462, October 2002.

[2] 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(1):36–56, 2005.

[3] E. Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs Paralleles*, 10(2):141–171, 1998.

[4] E. Cantú-Paz. Migration policies, selection pressure, and parallel evolutionary algorithms. *Journal of Heuristics*, 7:311–334, 2001.

[5] Francisco Fernández de Vega, Marco Tomassini, and Leonardo Vanneschi. Studying the influence of communication topology and migration on distributed genetic programming. In *EuroGP '01: Proceedings of the 4th European Conference on Genetic Programming*, pages 51–63, London, UK, 2001. Springer-Verlag.

[6] M. Ericsson, M. Resende, and P. Pardalos. A genetic algorithm for the weight setting problem in OSPF routing. *J. of Combinatorial Optimization*, 6:299–333, 2002.

[7] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *INFOCOM 2000*, pages 519–528, 2000.

[8] B. Fortz and M. Thorup. Increasing internet capacity using local search. *Computational Optimization and Applications*, 29(1):13–48, 2004.

[9] Tobias Grundmann, Marcus Ritt, and Wolfgang Rosenstiel. TPO++: An object-oriented message-passing library in C++. In David J. Lilja, editor, *Proceedings of the 2000 International Conference on Parallel Processing*, pages 43–50. IEEE Computer society, 2000.

[10] Patrick Heckeler, Marcus Ritt, Jörg Behrend, and Wolfgang Rosenstiel. Object-oriented message-passing in heterogeneous environments. In A. Lastovetsky et al., editors, *EuroPVM/MPI*, volume 5205 of *Lecture Notes in Computer Science*, pages 151–158. Springer, 2008.

[11] J. Holland. *Adaptation In Natural and Artificial Systems*. The University of Michigan Press, 1975.

[12] Ewing L. Lusk and Anthony Chan. Early experiments with the OpenMP/MPI hybrid programming model. In Rudolf Eigenmann and Bronis R. de Supinski, editors, *IWOMP*, volume 5004 of *Lecture Notes in Computer Science*, pages 36–47. Springer, 2008.

[13] M. Pioro M, A. Szentsi, J. Harmatos, A. Juttner, P. Gajownicczek, and S. Kozdrowski. On open shortest path first related network optimization problems. *Performance Evaluation*, 48(4):201–223, 2002.

[14] OpenMP Forum. OpenMP Application Program Interface. Technical report, May 2008.

[15] Roger Reis, Marcus Ritt, Luciana Buriol, and Mauricio G. C. Resende. A biased random-key genetic algorithm for ospf and deft routing to minimize network congestion. *International Transactions in Operational Research*, pages 401–423, 2011.

[16] Lorna Smith and Mark Bull. Development of mixed mode MPI/OpenMP applications. *Scientific Programming*, 9(2–3):6–7, 2001.

[17] State of Baden-Württemberg. bwGRid. `www.bw-grid.de`.

[18] R. Tanese. *Distributed Genetic Algorithms for Function Optimization*. PhD thesis, University of Michigan, 1989.

[19] A. Wächter and L. T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-sscale nonlinear programming. *Mathematical Programming*, 106(1):25–27, 2006.

[20] Dahai Xu, Mung Chiang, and Jennifer Rexford. DEFT: Distributed exponentially-weighted flow splitting. In *INFOCOM 2007*, pages 71–79, May 2007.

[21] Dahai Xu, Mung Chiang, and Jennifer Rexford. Peft: Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering. In *INFOCOM 2008*, 2008.