Pp. 181–187 in *Proceedings of the 3rd World Multiconference on Systemics, Cybernetics and Informatics (SCI'99)*, edited by M. Torres, B. Sanchez, S. Radhakrishan and R. Osers. International Institute of Information and Systemics, 1999.

An Empirical Evaluation of Stochastic Search Methods in Real-World Telecommunication Domains

Jussi Lahtinen, Petri Myllymäki, Tomi Silander, Henry Tirri, Hannes Wettig
Complex Systems Computation Group (CoSCo)
P.O.Box 26, Department of Computer Science
FIN-00014 University of Helsinki, Finland
http://www.cs.Helsinki.FI/research/cosco/
cosco@cs.Helsinki.FI

ABSTRACT

In the literature there exist several stochastic methods for solving NP-hard optimization problems approximatively. Examples of such algorithms include (in the order of increasing computational complexity) stochastic greedy search methods, simulated annealing, and genetic algorithms. In this paper we are interested in the problem, which one of these methods is likely to give best performance in practice, with respect to the computational effort required in applying the method. We study this problem empirically by selecting a set of stochastic algorithms with varying computational complexity, and by experimentally evaluating for each method how the goodness of the results achieved improves with increasing computational time. For the evaluation, we use two practical optimization problems closely related to real-world problems in telecommunications. To get a wider perspective of the quality of the results, the stochastic methods are also compared against special-case greedy heuristics. Our investigation indicates that although good results can be achieved by genetic algorithms, simpler stochastic algorithms can achieve similar performance with less computational resources. However, of the methods studied, genetic algorithms appear to be the most consistent in the sense that the variance in the quality of the results obtained is smallest with this approach.

1. INTRODUCTION

Real-world phenomena are complex, as indicated by increased interest in the "Science of complexity" [4]. It is also evident that the computational problems encountered in real environments differ from text-book optimization problems, which in many cases only serve as tractable examples to illustrate the nature of problems. In real-world applications one is faced with computational problems where finding the optimal solution requires (at least) an exponential number of steps with respect to the input instance (i.e., the problem is at least NP-hard). Since such problems most probably are not solvable efficiently by any deterministic method, we are forced to search for efficient methods producing good approximate solutions. Such approximate methods with performance guarantees are known for many classes of problems.

In addition to their inherent complex computational nature, real-world problems also present another type of complication. The very notion of optimality may be a complex entity. In many real cases the goodness (or cost) of the solution is not a simple function of some attributes of the solution, but rather a complex function of many different aspects of it. This makes it extremely difficult to develop heuristics for finding high quality approximate solutions for the problems. Difficulties in developing special heuristics have been the starting point for general global optimization methods for which the cost function appears essentially as a "black box". The interesting question is then, how this black box general approach performs with respect to the heuristic approximation algorithms. In other words, does neglecting information about the structure of the cost function yield essentially inferior optimization methods or do the additional benefits of general purpose methods (computation time, adaptability, ease of coding etc.) outweigh the possible loss in solution quality of "tailored" heuristic approaches.

The importance of other than pure solution quality factors in the evaluation of the approximation approaches is emphasized by the fact that in real life applications, the cost function's tendency to change during time is more a rule than an exception. This change is due to the fact that optimality is often partly defined by some real-world parameters, e.g., capacities and resource quantities. In addition, for some cases even structural changes to the function are possible, e.g., when fast evolving technology allows new types of modules to be added to the environment. Developing new special heuristics after each change in cost function parameters or function structure turns out to be in most cases a very inefficient approach, sometimes even infeasible. In such cases approximation methods that handle cost functions as black-boxes have a specific advantage over more tailored approaches, as the changes in the cost function only means changing or re-parameterization of the black-box.

In this work we are interested in the wide class of *stochastic optimization methods*, in particular such method families as simulated annealing [1] and genetic algorithms (see e.g. [6, 2]). Our interest in stochastic methods (in addition to their wide applicability and the related theoretical appeal) stems from the fact that in real-world problems, the formalized cost functions seldom succeed in defining precisely the real optimality of the solution. This real optimality condition is in many cases far too complex (and probably also ill-defined) to be grasped by any particular formula. Rather, the formalized cost function can be seen as an approximation of the real-cost function. In such cases it is desirable to have methods, which can suggest a set of good solutions from which the solution to be used can be chosen, i.e., allow "hedging one's bets". Due to the randomness involved, stochastic optimization methods demonstrate such a behavior. Needless to say, stochastic optimization methods handle cost functions as black-boxes, and thus can be considered as promising candidates for solving real-world optimization problems.

Although many stochastic optimization methods can be seen as general tools for any optimization problem, there is of course no guarantee that they are equally applicable with good performance over a set of problem domains. It also very difficult to categorize problem areas in such a way that one is able get a fair comparison of different algorithms. In addition, one would like the comparison problems to reflect some realistic application domain as artificial problems often allow simpler approaches than the real ones. In this paper we have chosen as our evaluation framework two problems in the telecommunications domain: a tree-problem, which is of practical importance in network design, and a frequency allocation problem important in mobile telecommunication networks.

From the plethora of stochastic algorithms we have chosen three well known, but interestingly different approaches. Firstly, we study the performance of so called *stochastic greedy heuristics*, which usually are to some extent customized to solve the specific problem at hand. A priori we expect such algorithms to run quickly, with the disadvantage of getting stuck at local minima. On the other hand, fast execution times allows us to perform several repetitive runs (initiating from different starting points), each leading to a possibly different good solution. Our second algorithm, *simulated annealing*, is a step towards more general purpose, but possibly slower methods. At the same time, it is a shift from pure Monte Carlo -approach towards more sophisticated stochastic algorithms. As such, it stands in a midway to our third family of algorithms, *genetic algorithms*, which a priori can be expected to demonstrate a very robust, but possibly very slow behavior.

The characterizations above should be regarded as generally held prior conceptions about these algorithms and serve in this study as working hypotheses. We are fully aware of the insufficiency of this kind of case study to validate any general claims. However, when carefully performed and thoroughly documented, comparing different approaches on several well chosen problems still gives us some useful evidence and further information on the relationship of the different approaches.

2. THE PROBLEMS

In our experiments reported in Section 4., we used two real-world problems which are both related to the telecommunication domain, but are of quite different nature. The first one is a *graph optimization problem* which deals with networks of V nodes, one of which is a special root node. The target is to find the most cost-effective way to connect the nodes by directed arcs so that the arcs form a tree $\mathcal{G} \subset V \times V$. The cost function for a tree $\mathcal{G} \subset V \times V$ is defined hierarchically by

$$\mathsf{Cost}(\mathcal{G}) = \sum_{v \in V} \mathsf{NodeCost}(v) + \sum_{e \in \mathcal{G}} \mathsf{ArcCost}(e) \qquad (1)$$

$$\mathsf{ArcCost}(e) = \mathsf{ArcPrice} \cdot |e| \cdot \left\lceil \frac{|\mathsf{Subtree}_{\mathcal{G}}(e)|}{\mathsf{ArcCapacity}} \right\rceil$$

$$\mathsf{NodeCost}(v) = \mathsf{ConnPrice} \cdot \left\lceil \frac{|v|}{\mathsf{ConnVolume}} \right\rceil$$

In our experiments, we have parameterized the cost function by using the following constants:

ArcPrice	10.0
ArcCapacity	8.0
ConnPrice	0.1
ConnVolume	2.0

It should be noted that as the cost function does not depend only on the length of the connecting arcs, the simple minimum spanning tree (MST) solution does not produce a good solution. More details of this problem domain can be found in [11].

The second problem studied in this paper is the *frequency allocation problem (FA)*, where the goal is to allocate the available channels to a number of transmitter nodes (TRXs) so that the the separation constraints are fulfilled, and the interference cost is minimized. The separation constraints are a set of requirements which give the minimum allowed distance between allocated channels of two TRXs. The cost function used in this study is a simplified version (no frequency hopping) of the cost function presented in [15].

It should be observed that these two problems are very different also in the sense that for the first one finding a valid solution is quite straight-forward — any complete tree with the specified root node as root is a solution — whereas in the second problem legal solutions are highly constrained, posing already initially a more difficult search problem.

3. THE ALGORITHMS

For solving the two optimization problem defined in the previous section, we experimented with several different variations of stochastic algorithms. The best results achieved were obtained with the algorithm instances shown below. To get a wider perspective of the results achieved, we also used Kruskal's deterministic algorithm [10] for generating a *minimum spanning tree (MST)* solution with respect to Euclidean metric of connecting arcs for the graph optimization problem. As the cost function (1) does not depend only on the Euclidean distances of the nodes, the MST algorithm does not produce an optimal solution to our optimization problem, but it serves as a useful reference point for our experiments with the stochastic algorithms.

3.1. Stochastic greedy algorithms (SG)

By *(local) greedy search* methods we mean in the sequel algorithms were the solution space is searched iteratively using small local modifications and the modifications are accepted only if the value of the cost function is decreased. The search is local in the sense that the modifications considered are such that only one component of the solution can be changed at each time step. *Stochastic greedy algorithms* are algorithms which iteratively start greedy searches from different starting locations.

With the graph optimization problem we ran the stochastic greedy algorithm using two different variations. The first, most greedy version (denoted in the sequel by *MST-SG*) uses the MST solution as the starting point, and exploits minimal size local neighborhood for generating new solution candidates. By minimal size local neighborhood we mean here that when an arc is (randomly) chosen to be the target for a local modification, only nodes directly connected to the end points of the arc are considered as the new end points of the modified arc. The second version (denoted in the sequel by *RST-SG*) uses random starting points and the largest possible local neighborhood, where any the node in the network can be the new end point for an arc to be modified.

In the FA problem domain, we found the standard stochastic greedy algorithm to work reasonably well. We also performed experiments with a *permutation-based stochastic greedy (PSG)* search method, were instead of changing the frequency of a randomly chosen TRX at a time (which is the case with the standard SG algorithm), the TRXs are optimized by using a greedy algorithm which goes through the TRXs in a specific order. The stochastic aspect of the algorithm lies in the choice of the order to be used: the algorithms starts with a random order, and makes then randomly small perturbations in the ordering accepting the changes greedily, i.e., only if the new order combined with the greedy search produces better results.

3.2. Simulated annealing (SA)

Simulated annealing (SA) [1] is a stochastic optimization method based on the theory Markov chain processes. In an SA process, the probability of accepting a move which leads to increased cost depends on a real-valued parameter called the *temperature*. It can be shown that if the temperature is first set to a high value, and then decreased (cooled down) slowly enough, the SA process will converge to the minimum cost solution with probability one [7]. Unfortunately, this interesting theoretical result requires an exponential number of iterations, but empirical results show that near-optimal minima can be obtained rather easily with a polynomial number of iterations.

In our experiments, parameters of simulated annealing were adjusted manually to perform well on different size problems. For the cooling schedule, we used a simple, but widely used scheme where the current temperature is decreased by multiplying it with a *cooling factor* CF, CF<1. At each temperature T, the SA updating process was iterated s (*sweepsize*) times before the temperature was decreased. In the results reported in the next section, the initial temperature was set to 100.0, final temperature was 0.01, sweepsize $2 \cdot |V|$, and the cooling factor 0.9817. To make the comparison with RST-SG easier we used the same maximal local neighborhood for generating new solution candidates.

With the FA problems tested, we obtained the best results by using 1.0 as the initial temperature, 0.00001 as the final temperature, 10 as the sweepsize, and 0.99 as the cooling factor, although with the largest problem instances encountered (with over 300 nodes), the initial temperature needed to be raised to 5, sweepsize to 30, and cooling factor to 0.999.

3.3. Genetic algorithm (GA)

Genetic algorithms (GA) are a relatively new family of stochastic optimization methods where the state of the search process at a certain time consists of a set (*population*) of solution vectors (*chromosomes*). For coding the solutions as chromosomes in the graph optimization problem, we used a scheme described in [11], which is a modification of the methods suggested originally in [13].

For implementing the GA in the graph optimization domain, we experimented with a large number of different parameter settings. The results reported in the next section were obtained by using a population of size 200, with crossover probability set to 0.80, mutation probability set to 0.01, and by running $2000|V|$ iterations with a graph with $|V|$ nodes. For the crossover operation, we found the uniform combination of the chromosomes to work best in this domain.

Implementing the GA in the frequency allocation domain turned out to be much more difficult than with the graph optimization problem. One reason for this is the fact that to successfully run a genetic algorithm we need a heterogeneous initial population of solutions. In very constrained problems finding even a single legal solution is hard (here NP-hard), thus finding a good heterogeneous population may be infeasible. Furthermore, defining a successful cross-over operation in constrained problems is generally a very tedious task. In the real-world frequency allocation problems studied in this work the separation constraints were found to be such that a properly working cross-over operation could not be found. These difficulties can be circumvented by using the indirect permutation-based search introduced for the PSG method, since in this case we may operate on permutations all of which always represent (through the permutation-dependent greedy search) a legal, locally optimal solution. For this reason, in the FA domain we implemented the GA only on the permutation string space (similarly to PSG) rather than directly on the actual solution space.

Finding suitable parameter values for the GA algorithm appeared to be a much more complicated issue than with the SA approach. A good choice for the selection probability proved to be the following, where p_i, the probability that the ith-worst string is chosen to couple, is determined as a function of the *rank* (see [3]) i:

$$p_i = \frac{\sum_{j=1}^{i} j^2}{\sum_{j=1}^{M} \sum_{k=1}^{j} k^2},$$

where M denotes the size of the population. This function was chosen to be quadratic since linear functions appeared to be too tolerant, letting too weak individuals to couple, and cubic functions led to a too elitist process that soon converged to a homogeneous uniformity. During the GA, the fittest individual was always saved as an extra member of the population. Before losing its status as the best individual, and thus becoming overwritten, the best individual could be mutated, or become a parent, just like any other individual.

In cross-over, the *cycle cross-over* operator (see [12]) worked best, although the *stem cross-over* operator also gave good results. In the stem cross-over the stem is taken up to a single cutting point from the first parent, and the missing numbers in the permutation string are then added in the order they appear in the second parent string. The *order cross-over* (see [5]) slowed the GA's convergence considerably and was not able to produce very good results. For the cross-over probability, value 0.7 seemed to be a good choice: values anywhere near this point produced similarly good results.

Of the mutation operators, the *switch* operation (which exchanges two random positions in the permutation string) turned out to work better than the *shift* or *translate* operations. The mutation probability parameter appeared to be less robust than the cross-over probability: if the value was too small, the individuals lost their diversity. Best results were obtained by setting the mutation probability to 0.9, in which case 90% of the individuals were mutated at once.

For the population size, about 50 seemed to be quite sufficient — more appeared to be a waste of computing time for the problem sizes used in the experiments (with less than 353 TRXs).

4. THE RESULTS

Experimental comparison of stochastic algorithms is a subtle issue due to the inevitable increase in the number of tests caused by the random factors involved (random seeds, operation probabilities etc.) Furthermore, in this study we are not only interested in the quality of the solution, but also the time spent to find that solution. Therefore instead of just running each algorithm a fixed number of times we have "normalized" the experimentation by giving a fixed time available for each method. This is achieved by first fixing the number of repetitive runs allowed for the genetic algorithm, and then allowing each algorithm to produce as many candidate solutions as the genetic algorithm during the given number runs. With this type of scaling we allow more trials for faster algorithms such as stochastic greedy search, as long as the number of generated candidate solutions does not exceed the candidates generated by the genetic algorithm. This experimental setting corresponds approximately to the real life situation, where a faster stochastic method can be run several times during the same time interval a slower method is executed only once. In most application domains one is interested in the trade-off between the quality of the solution and the time to find it, and the true pragmatic optimum is a balance between the solution optimality and the resources required.

4.1. The graph optimization problem

In our experimental setting the algorithms discussed in the previous section were tested with data consisting of different graph sizes between 25 and 75 nodes with 5 (4 in the 75-node case) randomly generated instances for each size. This additional test parameter (graph size) was included in order to study the differences in the algorithm behavior with respect to the size of the search space. If such a "scaling behavior" is not taken into account, experiments with small instance sizes will give misleading results for real life applications where scaling requirements are a necessity.

Representative examples of the combined results for all algorithms are presented in Figures 1 and 3. The y-axis is the quality of the solution, *averaged over the tests performed* (lower cost represents better quality), and the x-axis is the number of candidate solutions searched. The minimum spanning tree (MST) result is shown as a reference solution — obviously the solution quality of a deterministic algorithm does not change with time, so the behavior of the MST algorithm in these figures is a line.

Two general observations are in order, before we discuss each of the individual algorithms. Firstly, all the algorithms tried show the expected behavior with an initial phase, where the solution quality improves very fast when the number of candidates examined grows. The improvement in the solution quality is very minimal after the "knee point", which indicates that in practice the search could be stopped with a little loss in accuracy after the difference between the quality of the consecutive solutions is below a (problem-specific) threshold. Secondly, it is important to notice that all the stochastic algorithms used improve the heuristic solution considerably (approximately 30 %). This makes us cautious about the usual claims in the literature, which state that a particular stochastic algorithm family has "achieved drastic improvements in the solution quality with respect to the traditional approaches". Such statements, though true (as the solution quality has indeed improved), can easily be misleading by indicating that this improvement is resulting from particular complex details of the algorithm, although similar results could also be achieved with simpler stochastic approaches. Consequently, our experimental evaluation suggests that any stochastic algorithm of such a complexity as most genetic algorithms, should not only be compared to heuristic approaches, but be evaluated also against simpler stochastic approaches in the the same problem domain.

The fastest stochastic algorithms considered, the two variants of stochastic greedy search (MST-SG and RST-SG) both performed well for all graph sizes. They found acceptable solutions very fast, especially MST-SG, the variant that starts from the minimum spanning tree solution. For small graphs the random tree variant (RST-SG) found better solutions than MST-SG with its more focused search, but the situation was reversed for larger instances. This behavior is as expected, since when the search space grows, the sampling approach adopted in RST-SG is not able to cover large enough areas of the space to compensate for the focused search in a promising subspace performed by the MST-SG algorithm. Naturally, this behavior depends on the denseness property of the solutions, i.e., on the property that a good heuristically found solution is located in a subspace densely populated with even better solutions. However, this is likely to be the case in domains where for example metric distances are used as input parameters.

As indicated by the related theory, a properly implemented simulated annealing (SA) algorithm showed consistently good performance by producing the best solutions of all the algorithms included in this study. For small instances the behaviors of the genetic algorithm and the simulated annealing algorithm coincide, but the larger the instance size, the greater the difference. Albeit slow with respect to the stochastic greedy algorithms, the simulated annealing algorithm was still much faster than the genetic algorithm. The difference in the solution quality is substantial at the point where the improvement in solutions produced by the genetic algorithm is already very small, thus indicating that the advantage from the crossover operation in our experiments is only neglible. Again, although the results presented here do not necessarily generalize to all domains, it should be noted that if striving towards the best quality solution is the main goal with lots of CPU-time available, simulated annealing is a very viable alternative.

Finally, as expected, the genetic algorithm was the slowest in finding good solutions. A more surprising observation was that the difference to its close relative, simulated annealing was so substantial. It is evident that given enough time the slow improvement in solution quality closes the gap between simulated annealing and the genetic algorithm, but from the curves one can estimate that this takes a very long time. One common criticism of results like ours is that in genetic algorithms the number of parameters (population size, mutation and cross-over probabilities etc.) is much larger than in the alternative approaches, and thus the experimental results presented depend heavily on the success of choosing good parameter values. This is definitely the case, but does not remove the fact that parameter selection is also part of the problem, and in any given timeframe we have only a restricted number of trials that can be performed. In our domain, as in many others, the genetic algorithm approach does not offer any theoretically justified means to select proper parameters for the algorithm. Thus approaches of lesser complexity (and thus with lesser number of parameters) have the advantage over more complex ones. It seems that in our domain few

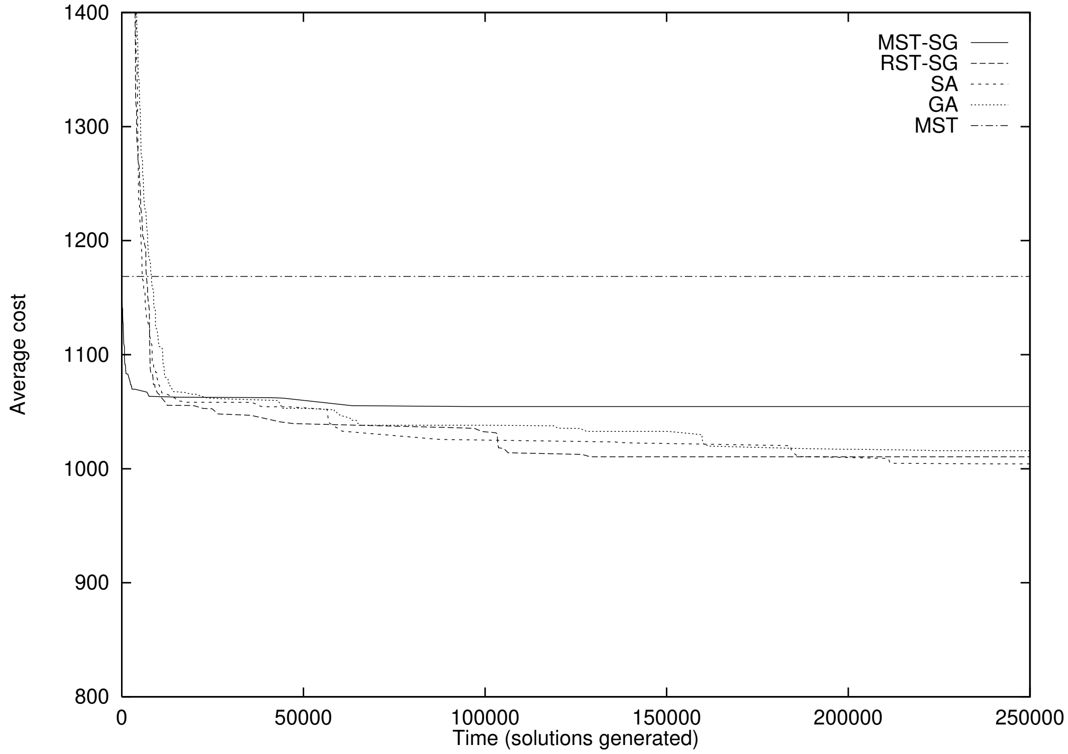

Figure 1: Empirical results with 25-node problems.

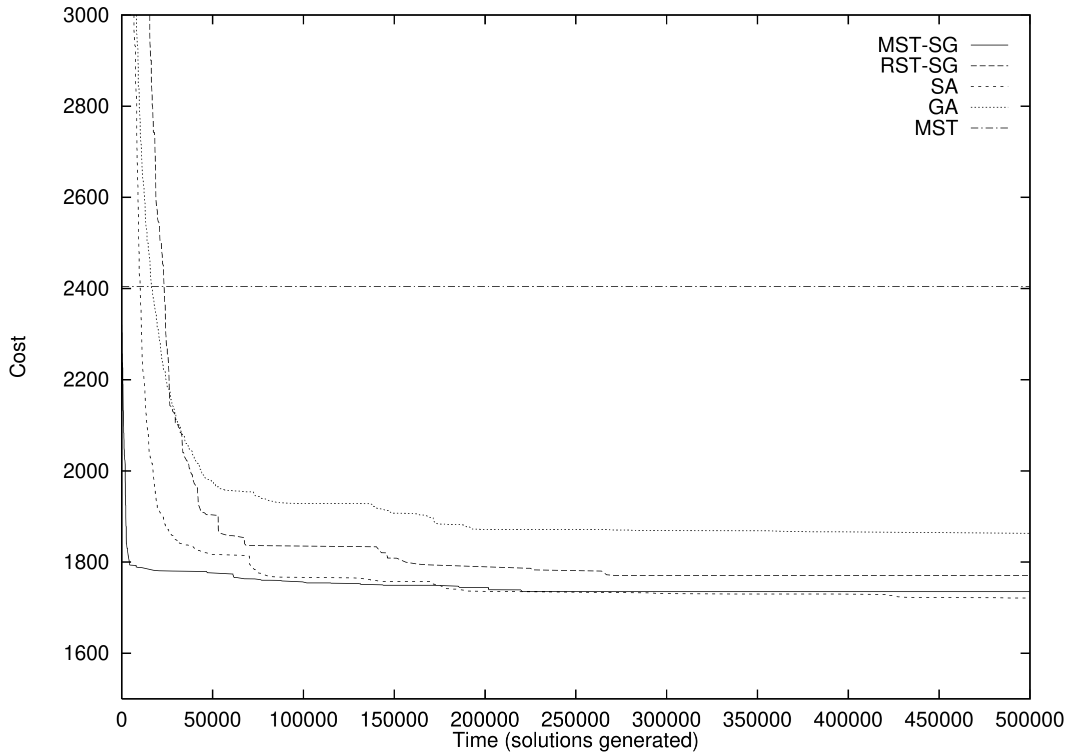

Figure 2: Empirical results with 50-node problems.

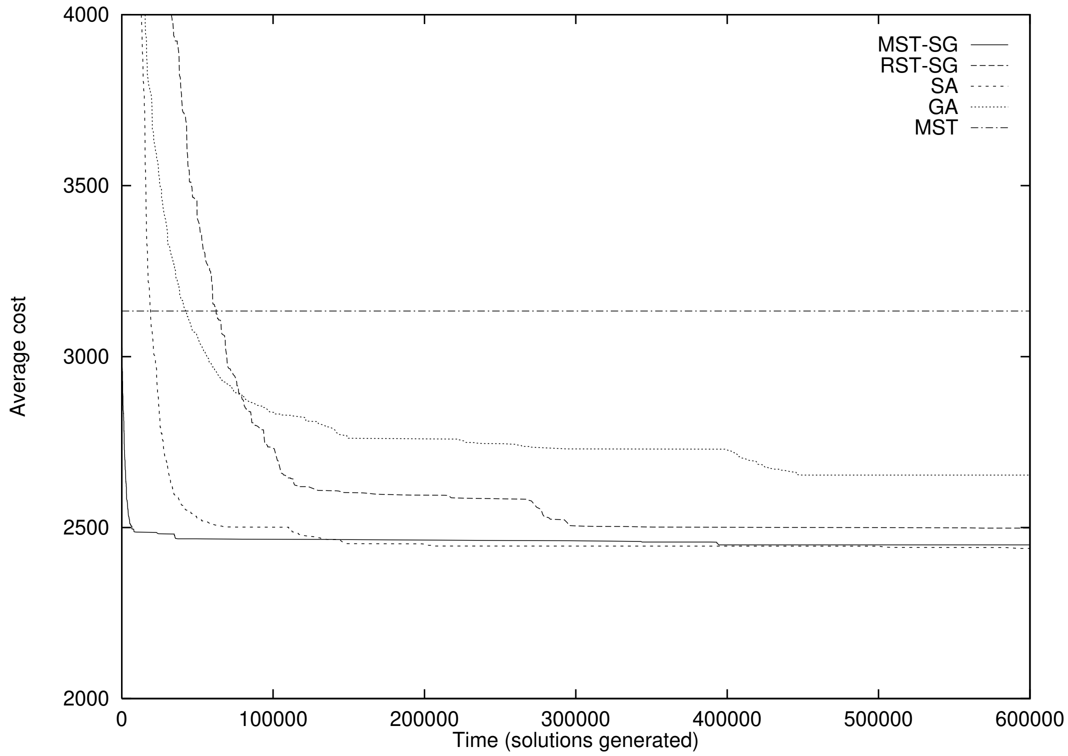

Figure 3: Empirical results with 75-node problems.

	SA	SG	PSG	GA
N	398	6837	40	38
min	**12.28**	14.84	13.74	14.09
max	**14.39**	18.84	15.29	15.43
avg	**13.48**	16.72	14.35	14.96
s	0.15	0.30	0.10	**0.08**

Table 1: Results with 352 TRX nodes, 8 channels, and loose separation constrains.

parameters regulating the stochastic search process are already sufficient for achieving good solutions.

4.2. The frequency allocation problem

Representative results with three different real-world frequency allocation problems of varying size and complexity can be found in Tables 1–3. In these tables, **N** denotes the number of iterative runs performed by using the algorithm is question, **min** the minimum of the cost function values obtained (the best result) during the N runs, **max** the maximum of the cost function values obtained (the worst result) during the N runs, **avg** the average of the cost function values obtained during the N runs, and **s** the standard deviation of the results obtained.

The number of iterative runs performed (N) was chosen in each case in such a way that the resulting total amount of computational resources required is approximately the same. Con-

	SA	SG	PSG	GA
N	397	1504	40	40
min	**84.83**	85.06	85.13	85.04
max	85.88	86.15	85.88	**85.32**
avg	85.30	85.49	85.33	**85.18**
s	0.03	0.02	0.02	**0.01**

Table 2: Results with 185 TRX nodes, 65 channels, and strict separation constrains.

	SA	SG	PSG	GA
N	398	996	40	40
min	**50.33**	50.92	51.14	50.82
max	52.52	53.25	52.65	**51.73**
avg	**51.30**	52.15	51.75	51.34
s	0.15	0.18	0.15	**0.04**

Table 3: Results with 202 TRX nodes, 26 channels, and strict separation constrains.

sequently, we can see that e.g., in a time needed for running the SA algorithm 398 times, the GA algorithm could be performed only 38 times. This means that with the parameters used here, a single run of the SA algorithm takes only about one tenth of the time required for completing one GA run. Similarly, we can see that the SG approach is several times faster than the SA method.

The results show that the best results with respect to the **min** score (the overall best solution found) was always obtained by the SA algorithm. However, as noted above, the SG algorithm is several times faster, so if time is an issue, then the this straightforward stochastic approaches could be the best alternative.

The performance of the algorithms is somewhat sensitive to the strictness of the constraints. Not surprisingly, the GA performs better in strictly constrained problems producing in some cases the best average and worst-case score. Part of this success is due to the indirect, permutation based coding of the problem. However, comparison with the PSG shows that the more sophisticated search of the GA outperforms PSG in strictly constrained problems. It should also be noted that as the standard deviation is quite small with the GA approach, the method is quite robust and does not usually produce very bad quality solutions at all.

5. CONCLUSIONS

In this paper we compared versions of three stochastic algorithms: stochastic greedy search, simulated annealing, and genetic algorithms. We investigated the question which group of methods is likely to give best performance in practice, with respect to the computational effort required. In order to answer this question, we presented an experimental setting where stochastic algorithms with varying computational complexity could be evaluated and compared. For the evaluation, we used two real-world optimization problems from the telecommunications problem domain. To get a wider perspective of the goodness of the achieved results, the stochastic methods were also compared against special-case greedy heuristics. The results suggest that although it is possible to get good results with genetic algorithms, a similar performance can be achieved faster by using stochastic algorithms with simpler structures, which supports the observations in [14, 9, 8]. Nevertheless, genetic algorithms appear to be quite consistent in the sense that variance in the quality of the results obtained is very small. The results of this work are currently being exploited in a major industrial software project.

ACKNOWLEDGMENTS

This research has been supported by the Technology Development Center (TEKES), Nokia Incorporated, and the Academy of Finland.

REFERENCES

[1] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. John Wiley & Sons, Chichester, 1989.

[2] J. Alander. An indexed bibliography of genetic algorithms. In L. Chambers, editor, *Practical Handbook of Genetic Algorithms: New Frontiers*, chapter Appendix 1, pages 333–427. CRC Press, Boca Raton, FL, 1995.

[3] J.E. Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms and their Applications*, pages 101–111, Pittsburgh, PA, July 1985. Lawrence Erlbaum Associates.

[4] J.L. Casti. *Complexification: Explaining a Paradoxical World through the Science of Surprise*. HarperCollins, New York, NY, 1994.

[5] L. Davis. Job shop scheduling with genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms and their Applications*, pages 136–140, Pittsburgh, PA, July 1985. Lawrence Erlbaum Associates.

[6] L. Davis. *Handbook of genetic algorithms*. Van Nostrand Reinhold, New York, NY, 1991.

[7] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

[8] L. Ingber and B. Rosen. Genetic algorithms and very fast simulated re-annealing: a comparison. *Mathematical and Computer Modelling*, 16(11):87–100, 1992.

[9] A. Juels and M. Wattenberg. Stochastic hillclimbing as a baseline method for evaluating genetic algorithms. Technical Report CSD-94-834, Department of Computer Science, University of California at Berkeley, 1994.

[10] J.B. Kruskal. On the shortest spanning subtree of a graph and the travelling salesman problem. *Proc. Amer. Math. Soc.*, 7(1):48–50, 1956.

[11] J. Lahtinen, P. Myllymäki, T. Silander, and H. Tirri. Empirical comparison of stochastic algorithms in a graph optimization problem. In J. Alander, editor, *Proceedings of the Second Nordic Workshop on Genetic Algorithms and their Applications*, pages 45–59, Vaasa, 1996. University of Vaasa and the Finnish Artificial Intelligence Society.

[12] I. Oliver, D. Smith, and J. Holland. A study of permutation crossover operators on traveling salesman problem. In *Proceedings of the 2nd International Conference on Genetic Algorithms and their Applications*, pages 224–230, Cambridge, MA, July 1987. Lawrence Erlbaum Associates.

[13] C. Palmer and A. Kershenbaum. An approach to a problem in network design using genetic algorithms. *Networks*, 26:151–163, 1995.

[14] K. Park and B. Carter. On the effectiveness of genetic search in combinatorial optimization. Technical Report BU-CS-94-010, Computer Science Department, Boston University, 1994.

[15] E. Sarkola. Channel assignment methods in frequency hopping cellular radio networks. In *Proceedings of the Eighth IEEE International Symposium on Personal, Indoor, and Mobile Radio Communications*, Helsinki, Finland, September 1997.