Local Search and Evolutionary Computation for Arc Routing in Garbage Collection

Thomas Bousonville

EuroBios
191, Av Aristide Briand
F-94234 Cachan Cedex
thomas.bousonville@eurobios.com

Abstract

The paper studies hybrid solution methods for a general class of arc routing problems arising in the context of garbage collection. Important differences of basic local optimizers for arc oriented compared to node oriented problems are worked out. The initial problem is split into its routing (sequencing) and clustering part. For both problems meta-procedures that make use of the modified local search procedures are proposed. The routing part is a well defined problem called Mixed Rural Postman Problem with Turn Penalties. For this problem an Evolutionary Algorithm is implemented and compared to known solution methods. It is able to provide the best known solution quality at the expense of high computational effort. The clustering part is shown to be very application dependent. To offer a flexible modeling of the problem a multi-agent-system using the formerly presented local search operators is proposed.

1 INTRODUCTION

The paper presents results from an applied research project where the computerized construction of new and the optimization of existing tours for inner-city garbage collection is investigated. As garbage collection is carried out along streets the problem is usually modeled as an arc routing problem. For reasons of restricted vehicle loading capacity and shift times the generic problem it refers to is called Capacitated Arc Routing Problem (CARP). Overviews of theoretic and application oriented work on this class of problems can be found in the surveys of Dror (2000), Assad and Golden (1995) and Eiselt at al. (1995). Necessary extensions to the standard CARP will be discussed below. Due to the complexity of the problem the solution process is normally split into a clustering and a routing step. The clusters then represent districts that are serviced by one vehicle on a specified day.

Some researchers have experimented with a 'route first-cluster second' approach, i.e. first building a 'giant' route through the hole collection area before partitioning it. This seems to be a promising approach if the total tour length is to be minimized. But it turned out that the staff responsible for planning is dealing with more complex objectives than minimizing the tour length (Bodin and Kursh 1978, Bodin et al. 1989). The additional requirements, besides a more detailed cost function (e.g. taking into account travel times which need not to be a simple function of length), concern the shape of the clusters. The resulting routes of the 'route first-cluster second' looked like pieces of threads, which was not considered to be a good result.

In this article both the clustering and routing problem will be addressed. For the latter an Evolutionary algorithm will be presented and its implementation will be compared to best known methods. For the clustering problem just a concept is proposed. From a methodological perspective it will be studied how local search can be used inside metaheuristics in order to solve both kinds of problems.

The outline of the paper is as follows. Chapter 2 gives a formal description of additional requirements for the routing problem without capacity constraints. Chapter 3 describes basic observations on local search procedures for this kind of arc oriented problems. Suitable local optimizers are then used inside an Evolutionary Algorithm for solving the problem defined in chapter 2. The description of the algorithm is followed by a summary of computational results and a comparison to alternative approaches. Chapter 4 deals with the district

planning problem and again gives ideas of how local optimizers can be embedded inside a higher level framework in order to yield a promising solution concept. The last chapter summarizes the findings and lists prospects for further research.

2 THE EXTENDED ROUTING PROBLEM

The routing task is a generalization of a common combinatorial problem: the Chinese Postman Problem (CPP). The CPP was first suggested by Guan (1962) and consists in finding a shortest route in a graph where every edge has to be visited at least once. Solutions to the CPP in polynomial time can be found for cases where the underlying graph is either undirected or directed. The mixed case was proven to be *NP*-hard and solution procedures have been proposed among others by Christofides et al. (1984).

In the garbage collection setting a network may contain one way streets and undirected streets. Furthermore an undirected street segment that does require separate service on each side will be transformed into two directed arcs instead of just one undirected edge. Therefore the underlying graph is of mixed nature. Consequently the problem is defined on a graph $G = G(N,E,A,c)$ with a node set N, undirected edges E, directed arcs A and a cost function $c:E \cup A \rightarrow \Re$. In the following elements of the unified set of both arcs and edges are called links and the associated set is denoted by $L = E \cup A$.

- We demand our model to cover turn restrictions or turn penalties as they exist at intersections. Turn restrictions imply that a path must not contain sequences

  $$(l_i, n_j, l_k)$$

  of a predefined set $T \subset (L \text{ x } N \text{ x } L)$ with $i,k \in \{1,..,|L|\}$, $j \in \{1,..,|N|\}$. Turn penalties have to be associated with extra cost. We model both requirements by introducing a function $p:T \rightarrow \Re$, which is taking a suitable large constant number if the turn is forbidden.
- Since service is required along links, but not necessary along all links a set $R \subset L$ defines the service links.

A CPP that does not require all arcs and edges to be included into the tour is commonly called Rural Postman Problem and therefore the problem is called Mixed Rural Postman Problem with turns penalties, MRPPTP (N, E, A, c, T, p, R). The MRPPTP has recently been formally defined and studied by Corberán et al. (2001). Before presenting the configuration of the developed Evolutionary Algorithm itself first some necessary modifications of standard local search operators will be discussed.

3 LOCAL SEARCH FOR ARC ORIENTED PROBLEMS

Research in the last decade has shown that the hybridization of generic concepts such as Genetic Algorithms can lead to enormous gains in solution quality for combinatorial optimization problems. In the case of evolutionary computing this has been demonstrated first for the Travelling Salesman Problem (TSP) (Mühlenbein et al. 1988). Genetic or Evolutionary Algorithms that incorporate problem specific knowledge in the form of local optimizers are called hybrid or memetic algorithms. The first naming is attributed to Goldberg (1989) whereas the word 'memetic' has been introduced by Moscato (1989) following an analogy to evolution in social systems. Many researchers have examined a variety of successful combinations of local search and population based approaches on the TSP (e.g. Merz and Freisleben 1997). The importance of the TSP for the considerd Arc Routing Problem stems from its similarity: both are ordering problems.

Popular local search procedures for ordering problems are 2-Opt, 3-Opt and the Lin-Kernighan operator. Transferring an approach for the node oriented problem *P1* to a arc routing problem *P2*, the following change of perspective occurs: The role of a node in *P1* is now taken by a link in *P2* and instead of edges between nodes in *P1* we now have to deal with (shortest) paths between links in *P2*. If two required links are directly adjacent in a tour the connecting path is obviously empty. Note, that if this was the case for all required links, the underlying graph would be eulerian, which does not constitute not a general property of the studied application. The change from an node oriented view to an arc oriented view inserts additional degrees of freedom in constructing the neighborhood, as will be seen below.

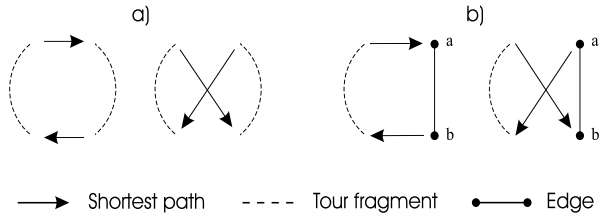

Figure 1: a) 2-Opt and b) Dir-Opt environment

The required links as well as the connecting shortest paths have a logical direction along the tour. Some of them (subset E) may be turned but others (subset A) may not. As a 2-Opt move (Figure 1a) changes the logical direction of one of the two involved sub-tours this approach can result in infeasible solutions. It would consume much

computation time to check the feasibility of every 2-Opt move at the innermost part of the algorithm. Additionally, if there is not a very small number of directed arcs in *L*, almost every sub-tour will be infeasible. Thus a high effort for a small chance of improvement would be undertaken. For this reason 2-Opt is not a good local optimizer for directed or mixed arc routing problems. Consequently the Lin-Kernighan operator which is immanently using a 2-Opt local search is not suitable either.

These observations for a 2-Opt environment are not specific to arc based problems but also hold for asymmetric node oriented problems. However a real difference and additional degree of freedom for arc based problems comes from the simple fact that $(a,b) \neq (b,a)$ whereas a node has no logical direction. As a consequence the classical 2-Opt environment can be extended by a move, which is illustrated by Figure 1b). This alternative does not exist for node oriented problems. The move in figure Figure 1b) can be viewed as a redirection of one service edge. In order to avoid confusion with the classical 2-Opt terminology we call the local optimizer based on this move *Dir-Opt*.

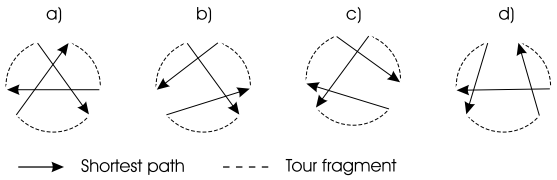

Figure 2: Four possibilities for a full 3-opt move

Now, let's consider the 3-Opt approach. A 3-Opt move is made by first removing three connecting shortest paths from the tour and then reconnecting the resulting three tour fragments in an optimal way. There are exactly sixteen possibilities to do that (including the original tour). As shown above 3-Opt moves which in fact correspond to a 2-Opt move turn at least the direction of one tour fragment. The same is true for 3-Opt moves that correspond to an "extended" 2-Opt move (i.e. they replace one of the removed shortest paths by simply redirecting it). Figure 2 enumerates all 3-Opt moves that do not turn the direction of the upper left tour fragment and constitute neither a 2-Opt move nor an extended 2-Opt move.

From Figure 2 it is easy to see that there is only one 3-Opt alternative to the current tour which maintains the logical direction of all arcs and shortest paths in the tour fragments. Figure 2a) is the only practically feasible exchange step, because turning the direction of partial tours is costly and usually not possible for the same reasons as with 2-Opt.

In analogy to 2-Opt, the examination of a reversed sub-tour consisting of only one link is not that costly. In addition, the chance of being able to turn a single link is also bigger than for longer fragments. In the following a *3-OptS* move will denote the exchange of three shortest paths without turning the direction of any of the tour fragments except if this tour fragment consists of only one link.

4 AN EVOLUTIONARY ALGORITHM FOR THE ROUTING PROBLEM

4.1 THE EVOLUTIONARY FRAMEWORK

Representation and objective function

Evolutionary computing has proved to be able to provide good solutions for hard combinatorial problems. Research on evolutionary algorithms for ordering problems has been exhaustive.

The first and most important choice to be made when designing an EA is how to represent the problem. In the literature different representations have been studied and the path representation has become generally accepted. In the case of the MRPPTP this means that

$$ind1 = \left((r_1, d_1),..,(r_{|R|}, d_{|R|})\right)$$

represents a solution which contains the service links in the order r_1 followed by r_2 and so on. The binary variable d_i indicates the direction in which the service link r_i has to be traversed.

To reconstruct the tour, r_i and r_{i+1} are connected by their shortest path. Paths that take into account turn penalties are also referred to as feasible chains (Benavent and Soler 1999). A feasible chain from link $l_1 := r_i$ to link $l_k := r_{i+1}$ is an alternate sequence of links and turns $C = \{l_1, t_1, .., l_{k-1}, t_{k-1}\}$ with $t_i = (l_i, n, l_{i+1})$ where n is a node shared by l_i and l_{i+1}. Using this notation the cost for a feasible chain sums up to

$$c(C) = \sum_{i=1}^{k-1} c(l_i) + p(t_i)$$

A shortest feasible chain $C^{\min}_{k,k+1}$ from (r_k,d_k) to (r_{k+1},d_{k+1}) is consequently a feasible chain from r_k (having direction d_k) to r_{k+1} (having direction d_{k+1}) that computes to minimum cost. Now we can formulate the objective function for the MRPPTP:

$$\sum_{k=1}^{|R|-1}\left(c(C^{\min}_{k,k+1})\right) + c(C^{\min}_{|R|,1}) \rightarrow \min$$

The path representation has a big advantage. The turn restrictions set out in section 2 can be included into the calculation of the shortest paths between the two service edges. This calculation is polynomial and has to be done only once before a EA run. It is not possible to apply a normal Dijkstra algorithm, because in graphs with turn penalties a node may occur more than once along a

shortest path. This is not the case for links. Each link can occur only once along a shortest path with turn penalties. This observation leads to a modified version of the Dijkstra algorithm, in which links are scanned instead of nodes. This procedure has a complexity of $O(|L|^2)$.

Operators and population management

Based on the path representation operators like the order crossover OX (Davis 1985), the partially mapped crossover PMX (Goldberg and Lingle 1985) and the edge (here: shortest path) based DPX operator (Merz and Freisleben 1997) can be applied. The PMX operator holds its merits mainly for problem classes where the absolute position of a gene is of relevance. It is therefore not surprising that DPX and OX led to better solutions. Computations further revealed the superiority of the OX. Figure 3 shows a typical development of the mean values of three runs of each combination for a given instance. In addition to crossover we define a mutation operator that simply exchanges the positions of two service links within the string, but not their directions.

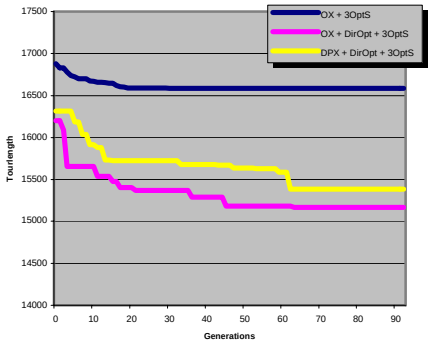

Figure 3: Mean development of different configurations

The selection process is done by choosing *m* parents for mutation (mutation rate = *m*/*popSize*) and *c* parents for crossover (crossover rate = *c*/*popSize*) independently using the stochastic universal sampling method (Baker 1987). In this method the probability of an individual of being selected for either crossover or mutation is proportional to its relative fitness. There is no risk of a dominant super-individual because the local search step preceding the evaluation evens out dramatic differences in fitness among the individuals.

The resulting *m*+*c* offsprings are copied to the new population. The rest of the new population (*popSize*-*m*-*c*) is filled with the fittest individuals from the parent population. This means that the population size stays constant over all generations. As long as *c*+*m*<*popSize* it is also guarantied that the best individual of the parent population is kept for the new population.

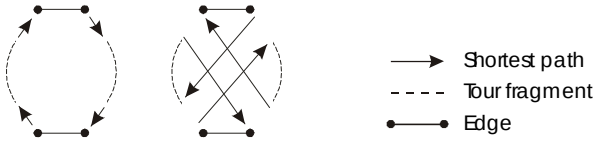

Figure 4: Mutation operator from a neighbourhood perspective

When designing hybrid evolutionary algorithms incorporating local search one has to take care that the local optimizer does not systematically undo the changes made by the EA operators. For the OX-crossover this is obviously not the case. Figure 4 visualizes the work of the mutation operator. Exchanging two service links in the path representation is equivalent to a 4-Opt move. The possibility that several subsequent 3-OptS moves undo such a move is very small.

In figure 5 the outline of the algorithm is given. Initialization of the individuals is done with a random permutation of the required links and their direction.

```
procedure MRPPTP_EALS
begin
    t := 0
    init population P_t of size n
    for each individual i∈P_t do
        DirOpt (i)
        3-OptS (i)
    end
    while not converged do
        evaluate all i∈P_t
        select m parents for mutation
        select c/2 parent pairs for
          crossover
        copy the resulting offsprings
          to P_t+1
        copy the n-m-c best individuals
          offsprings to P_t+1
        t := t+1
        for each individual i∈P_t do
            DirOpt (i)
            3-OptS (i)
        end
    end
end
```

Figure 5: Pseudocode of the algorithm

4.2 COMPUTATIONAL RESULTS

The paper of Corberán et al. (2001) on the MRPPTP presents complexity results and a transformation of the problem to the Asymmetric Traveling Salesman Problem (ATSP). Consequently they apply a known exact and a heuristic procedure (Patching heuristic, Karp 1979) to the ATSP. Additionally the authors develop a multi-stage problem-specific heuristic making use of Tabu Search elements at one stage, and which will be denoted by TS in the sequel. The neighborhood for this Tabu Search step is not the same as the one presented in section 3 and a direct comparison of the merits of the metaheuristics is not possible.

Corberán et al. (2001) constructed several benchmark sets. They are based on 18 computer generated base graphs from which variants are derived by adding additional arcs and edges. The variants are systematically derived only for 7 of the 18 base graphs and the 63 (7x3x3) variants of these 7 base graphs have been chosen for comparison here. They are representing the complete range of problem sizes and ratios of edges and arcs to be serviced.

For each instance six runs of the EA were performed, three runs with population sizes of 20 and 50 respectively. The exact ATSP algorithm was able to solve 24 of the instances. For these the EA found the optimal solution in 15 cases, for the remaining 9 problems the best found solution did not differ by more than 0.3% from the optimum. Over all instances the algorithm yielded tours which were in average 1% shorter than those gained by the TS approach. As can be seen from Table 1, column 3, the relative advantage is much bigger for smaller instances (up to 5.6%). On the other hand the TS procedure terminated usually in less than one minute whereas the EA needed between several minutes and 10 hours for a single run, depending on problem size. This time is nearly exclusively used by the local search operators which are superlinear in problem size.

The last two columns of Table 1 tell something about the robustness of the presented EA. The column titled "MeanEA/BestEA" shows, that the average result of an EA run is about 0.5% worse compared to the best result. The last column gives insight in the dependence of the solution quality on the population size. There is no dramatic loss in solution quality but the strategic aspect of the problem might justify the additional effort.

poblem size (|R|)	number of instances	TS/EA	MeanEA / BestEA	BestEA20 / BestEA50
1-99	2	1,056	1,000	1,000
100-199	17	1,017	1,004	1,002
200-299	21	1,008	1,006	1,003
300-399	14	1,005	1,006	1,003
400-499	7	1,002	1,005	1,002
500-599	2	1,002	1,007	1,003

Table 1: Summary of computational results

5 AN APPROACH FOR CLUSTERING

5.1 OBJECTIVES

When partitioning the whole collection area into clusters their distance to the depot and the landfill will have an influence on their shape and size. The type of the vehicle that is assigned to a cluster will also restrict the size of the cluster. But it is not only the difference in capacity that matters. Vehicle types may vary in tip technology and the size of the loading crew. This determines their speed during collection and their cost per time (due to higher amortization or personal costs). The single streets or sub-areas may possess attributes (e.g. bin per meter, settlement structure) that favor a vehicle type instead of another. Given the NP-hardness of the connected routing task inside each cluster, it seems hopeless to find an optimal partition of the collection area for an heterogeneous vehicle fleet.

The second point why global optimization falls short is the multidimensionality of the objective function. In the introduction it has been pointed out, that the planning team has additional optimization criteria in mind than minimizing tour length or even costs. A desired property of the clusters besides a balanced load is their *compactness*. What this colloquial term means in a graph-theoretic context is illustrated in Figure 6.

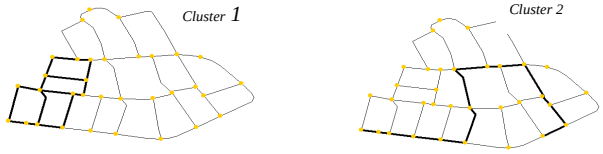

Figure 6: Two clusters (bold lines) varying in their degree of compactness

How can this intuitive idea be expressed by a formal measure? Clusters of the shape of Cluster 1 have a relatively small number of adjacent but not contained edges compared to the number of edges belonging to the cluster. This can be measured as

$$q = \frac{number\ of\ edges\ inside\ the\ cluster}{number\ of\ edges\ inside\ the\ cluster + number\ of\ edges\ adjacent\ to\ the\ cluster}$$

with $0<q\leq 1$. The measure q for a cluster can be easily calculated. Let M be the set of edges in the cluster and N^M the set of nodes induced by M. Then

$$q(M) = \frac{|M|}{\sum_{v \in N^M} (\deg ree(v)) - |M|}$$

The desired shape is only one example for the complex composition of the real-world objective function. The objectives can be even more subtly differentiated if e.g. time dependencies for the collection of special areas (pedestrian zones during shopping time, main roads during rush hours etc.) are introduced.

5.2 A MULTI-AGENT-SYSTEM APPROACH

These two observations, the individual conditions for each cluster and the complex objective function, lead to a higher level modeling perspective: view the vehicles as active parts of the optimizing system, let them be agents (Jennings and Wooldridge 1998). Every agent then represents a cluster, has the ability to communicate with other agents, i.e. tries to get, lose or exchange service links in order to increase its private objective function. An important advantage for the practical design of such a multi-agent-system (MAS) is that every agent can autonomously evaluate its fitness (degree of goal achievement).

To increase their fitness agents will interact among each other. This can be organized via a blackboard or in direct communication depending on the systems architecture. The question is: how does an agent determine efficiently whether or not an offered edge can increase its fitness? A complete run of the EA presented in the previous chapter would take to much time.

In this situation a locally optimal insertion of the edge can be computed and evaluated quickly. The local optimizers can also be used to determine the best edge to get rid off or to be replaced. The local operators as presented in chapter 3 can be embodied inside a multi-agent-system to generate quick responses in trading situations.

When the agent is idle (not in communication) a cluster optimization could be applied. How this is done depends on the problem representation inside the agent. It is possible that different agents use different optimizers depending on the structure and size of the clusters they represent.

For an implementation of this MAS further questions would have to be addressed: e.g. how to determine the number of agents. One could start with a heuristically calculated number and then, following the mean capacity usage after a certain time, merge or split agents. The system dynamics is flexible enough to handle a dynamic change in the number of agents.

Finally it is the general flexibility arising from the distributed, object-oriented modeling approach that represents the major advantage and appeal of the MAS.

6 CONCLUSION

The paper formalizes a general routing problem arising as part of a real world application. The specific situation of arc oriented compared to node oriented problems with respect to the design of effective and efficient local optimization techniques is studied.

The conceived local optimizers are applied successfully inside an Evolutionary Algorithm framework to solve the Mixed Rural Postman Problem with Turn Penalties. The results gained from this approach are the best known in terms of solution quality. In the examined application the algorithm was used for strategic decision support and quick computation was not an important factor. However the long running times, especially for bigger problems, may be a drawback for its application on other problems.

As a second problem the formation of clusters was presented. The individual and complex requirements for each cluster led to the idea of a multi-agent-system to model a distributed solution finding process. It is argued that the local search operators designed for the routing problem can constitute an important part of the interaction scheme of an agent. Prospects for further research include the implementation of this approach and a practical assessment of its performance.

Acknowledgements

The work has been financially supported by ISP-Project No. 6 of the Forschungsverbund Logistik of the University of Bremen.

References

Assad, A.A. and Golden, B.L. (1995) Arc routing methods and applications. In: Ball, M.O., Magnanti, T.L., Monma, C.L., Nemhauser, G.L., (eds.) *Network Routing*, pp. 375-483. Amsterdam: North-Holland.

Baker, J.E. (1987) Reducing Bias and Inefficiency in the Selection Algorithm. In: Grefenstte, J.J., (ed.) *Second International Conference on Genetic Algorithms,* pp. 14-21. Hillsdale, NJ: Lawrence Erlbaum Associates.

Benavent, E. and Soler, D. (1999) The Directed Rural Postman Problem with Turn Penalties. *Transportation Science* **33,** 408-418.

Bodin, L., Gagin, G., Welebny, R. und Greenberg, J. (1989) The Design of a Computerized Sanitation Vehicle Routing and Scheduling System for the Town of Oyster Bay, New York. *Computers and Operations Research* **16,** 45-54.

Bodin, L.D. and Kursh, S.J. (1979) A detailed description for the routing and scheduling of street sweepers. *Computers and Operations Research* **6,** 181-198.

Christofides, N., Campos, V., Corberán, A. and Mota, E. (1984) An optimal method for the mixed postman problem. In: P. Thoft-Christensen, (ed.) *System Modeling and Optimization. Lecture Notes in Control and Information Sciences 59,* pp. 641-649. Berlin, Heidelberg, New York: Springer.

Corberán, A., Martí, R., Martínez, E. and Soler, D. (2001) The Rural Postman Problem on Mixed Graphs with Turn Penalties. *Computers and Operations Research* . forthcoming

Davis L. (1985) Applying Adaptive Algorithms to Epistatic Domains. In: Joshi Aravind, (ed.) *Proceedings of the International Conference on Artificial Intelligence*, pp. 162-164. Los Altos, California: Morgan Kaufmann Publishers.

Dror, M. (2000) Arc routing problems: theory, solutions, and applications (ed.). Boston: Kluwer.

Eiselt, H.A., Gendreau, M. and Laporte, G. (1995) Arc Routing Problems, Part II: The Rural Postman Problem. *Operations Research* **43,** 399-414.

Goldberg D. E. (1989) *Genetic algorithms in search, optimization and machine learning*, Reading, Mass.: Addison-Wesley.

Goldberg, D.E. and Lingle, R. (1985) Allels, Loci and the TSP. In: Grefenstette J. J, (ed.) *Proceedings of the First International Conference on Genetic Algorithms,* pp. 154-159. Hillsdale, NJ: Lawrence Erlbaum Associates.

Guan, M. (1962) Graphic Programming Using Odd and Even Points. *Chinese Mathematics* **1,** 273-277.

Jennings, N.R. and Wooldridge, M.J. (1998) Applications of Intelligent Agents. In: Jennings, N.R. and Wooldridge, M.J. (eds.) *Agent Technology: Foundation, Applications and Markets,* New York et al.: Springer.

Karp, R.M. (1979) Patching Algorithm for the Nonsymmetric Traveling Salesman Problem. *SIAM Journal on Computing* **8**, 561-573

Merz, P. and Freisleben, B. (1997) Genetic Local Search for the TSP: New Results. In: *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation,* pp. 159-164. IEEE Press.

Moscato, P. (1989) *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms,* Pasadena, CA: California Institute of Technology, C3P Report 826.

Mühlenbein, H., Gorges-Schleuter, M. and Krämer, O. (1988) Evolution Algorithms in Combinatorial Optimization. *Parallel Computing* **7,** 65-85.

Finding Worst-Case Flexible Schedules using Coevolution

Mikkel T. Jensen
Department of Computer Science, University of Aarhus, Denmark.
email: mjensen@daimi.au.dk, http: www.daimi.au.dk/~mjensen/

Abstract

Finding flexible schedules is important to industry, since in many environments changes such as machine breakdowns or the appearance of new jobs can happen at short notice. In this paper a minimax formulation is used to develop a coevolutionary algorithm for finding worst case flexible schedules. A population of schedules is used to locate the schedule with the best worst case performance, while a population of breakdowns is used to locate the worst breakdown and estimate the performance of the schedules. This approach is compared to a standard scheduling approach and concluded to produce more flexible schedules. It is also compared to an approach in which the schedules are tested against all possible breakdowns; the coevolutionary approach is found to be faster and produce schedules of a comparable quality.

1 Introduction

Efficient scheduling is very important to industry, since it offers the promise of saving huge amounts of money by efficient use of resources. Most traditional research on scheduling has been focused on solving static problems in which every aspect of the problem is known beforehand, and in which nothing unforeseen ever happens. Recent research has focused on finding schedules that take into account possible future events. This has been done by creating *robust* schedules (schedules that are acceptable without a change if something unforeseen happens), [Her99, Jen01a, KY97, LWS94], or *flexible* schedules (schedules that are changeable to an acceptable schedule if something unforeseen happens), [BM00, Jen01a].

In the present paper the problem of finding worst case flexible job shop schedules is considered using a minimax formulation. During execution the schedules are facing machine breakdowns, after which rescheduling is done using a hillclimber. The schedules sought are supposed to have the lowest possible cost (makespan) after rescheduling for the worst possible breakdown.

The most straight-forward way of solving this problem is by evaluating the worst case performance of schedules by testing them against all possible breakdowns. Since there are many possible breakdowns, this approach is expected to be expensive in terms of processing time. Because of this, a more efficient way of testing the schedules is needed.

The algorithm presented is a coevolutionary genetic algorithm, in which a population of schedules coevolves with a population of breakdowns. The schedule population is evolved to minimise the worst case schedule cost after rescheduling of the breakdowns in the breakdown population, while the breakdown population is evolved to maximise the cost of the schedules in the schedule population after rescheduling. The breakdown population is used to estimate the worst case performance of the schedule population, which is expected to converge on the most flexible schedule.

This approach is compared to a genetic algorithm (GA) using a standard scheduling approach minimising schedule cost without breakdowns. It is also compared to a GA in which an exact evaluation of the worst case performance of the schedules is used.

The outline of the paper is as follows. In the next section job shop scheduling, rescheduling and breakdowns are introduced. Section 3 describes coevolutionary approaches to solve minimax problems. The three scheduling algorithms are described in section 4. The experiments and their results are discussed in section 5. The paper is concluded in section 6.

2 Job shop scheduling

A job shop problem of size $n \times m$ consists of n jobs $J = \{J_i\}$ and m machines $M = \{M_j\}$. For each job J_i a sequence of k_i operations $O_i = (o_{i1}, o_{i2}, \cdots, o_{ik_i})$ describing the processing order of the operations of J_i is given. Each operation o_{ij} is to be processed on a certain machine and has a processing time τ_{ij}. Sometimes each job is given a *release time* prior to which no processing of the job can take place. In the same way, sometimes each machine is given an *initial setup time*, prior to which no processing can be done on the machine. The scheduler has to decide when to process each of the operations, satisfying the constraints that no machine can process more than one operation at a time, and no job can have more than one operation processed at at time. Furthermore, there can be no *preemption*; once an operation has started processing it must run until it is complete.

Several performance measures exist for job shop problems. The performance measure used in this paper is the *makespan*, the time elapsed from the beginning of processing until the last operation has finished. The makespan is to be minimised.

The problem formulation above is a static definition; nothing unforeseen ever happens during the processing of a schedule. Real life scheduling is not like that. In the real world machines break down, deliveries get delayed, workers get sick and new jobs arrive during processing. In the rest of the paper, unforeseen events in the form of breakdowns will be considered. Here a breakdown is the temporary unavailability of a machine.

When an unforeseen event makes a schedule outdated the scheduler is faced with a rescheduling problem: find a new schedule incorporating the changes in the environment while respecting the part of the schedule already implemented. There are different ways of solving rescheduling problems. Since a rescheduling problem is a job shop problem, it can be solved in exactly the same way the preschedule (the schedule as it looked before the breakdown) was found. It is also possible to make use of the preschedule during rescheduling. The simplest kind of rescheduling is known as *right-shifting*; the processing order of the preschedule is kept, you simply wait for the breakdown to be repaired and then carry on with processing. A more efficient kind of rescheduling using the preschedule is hillclimbing; finding the new schedule by running a hillclimber on the preschedule.

Since the difficulty of a rescheduling problem depends on the preschedule, it is natural to take into account possible future events already when the preschedule is generated in order to guarantee some level of performance if something unforeseen happens. A schedule that is expected to perform well after unforeseen events and right-shifting rescheduling is usually termed *robust*, while a schedule expected to perform well after an unforeseen event and rescheduling using search is termed *flexible*.

There are different ways of formulating the expectation of performance after breakdowns. Two fundamentally different approaches are *average performance*, as considered in [Jen01a, LWS94], and *worst case performance* as considered in [KY97] and this paper. Furthermore, several kinds of worst case performance exist. *Absolute worst case performance* means minimising the cost of the worst case scenario, i.e., minimising

$$\varphi(x) = \max_{s \in S} F(x, s) \quad \text{subject to } x \in X$$

where $F(x, s)$ is the cost of schedule x after rescheduling breakdown s, X is the set of preschedules and S is the set of breakdowns. Using this kind of performance leads to a guarantee that no matter what breakdown happens, the actual cost will never be higher than $\varphi(x)$. Absolute performance focuses the scheduling on minimising the cost of the worst possible conditions. Another kind of worst case performance is *deviation worst case performance*, where the task is to minimise

$$\psi(x) = \max_{s \in S} [F(x, s) - F^*(s)] \quad \text{subject to } x \in X,$$

where $F^*(s) = \min_{x \in X} F(x, s)$ is the minimum cost achievable when scenario s happens. Relative worst case performance leads to a guarantee that no matter what scenario happens, the cost difference between the schedule optimal for that scenario and the schedule implemented will not be larger than $\psi(x)$. In this way relative performance focuses on finding a schedule that is always close to the best possible schedule for every scenario.

Absolute performance has the disadvantage compared to deviation performance that it can be necessary to exclude from S scenarios that cannot be countered by any schedule, otherwise absolute performance can turn out to be equivalent to standard static scheduling. However, since deviation performance has the added computational requirement that knowledge of $F^*(s)$ is needed for all $s \in S$, this paper focuses on absolute performance.

The rescheduling problems used in this paper are designed to resemble machine breakdowns. A machine breaks at a specific time (the *breakdown time*) and is in-operational for a certain time (the *down-time*), after which it comes back into service. If the machine is

processing an operation when it breaks down, the processing of this operation is delayed by the downtime. When the breakdown happens the scheduler is free to reschedule all operations that commence processing at the breakdown time or later in the preschedule.

In some situations a scheduler will be interested in making schedules flexible or robust to a particular kind of breakdown. Maybe one machine is particularly prone to breaking down, or maybe breakdowns tend to happen at specific times. Another reason to limit the breakdowns considered may be to exclude breakdowns that are known beforehand to be impossible to counter. If a specific machine is known to be the bottleneck at a specific time for all acceptable schedules, there is no way to generate robust or flexible schedules guarding against breakdowns of that particular machine at that time. In the same way, if a breakdown happens just before the end of all processing, it will not be possible to change the schedule, since there is no schedule left to change. It may make sense to exclude such machines and breakdown times from consideration in order to be able to find schedules that can cope with breakdowns of other machines.

In this paper a set of breakdowns B is characterised by a set of machines $B_M \subseteq \{M_j\}$, an interval of allowed breakdown times $B_T = \{T_{min}, T_{min}+1, \ldots, T_{max}\}$ and a downtime τ_B. Since a breakdown with a large downtime τ_B will always have more impact on the schedule than the same breakdown (same machine and breakdown time) with a smaller downtime, and since we are concerned with worst case performance, there is no need to vary the downtimes in a breakdown set.

The rescheduling problems used in the experiments were created from the preschedule s, the original problem, the machine breaking down m, the breakdown time T and the downtime τ in the following way:
1. The rescheduling problem is set to an empty problem with the same number of machines and jobs as the original problem.
2. All operations in the original problem with starting time of T or later in s are included in the rescheduling problem.
3. If an operation o is being processed on m at T, the release time of the job J_o that o belongs to is set to $\max(T, t_o) + \tau$, where t_o is the end of processing time of o in s. The release-time of any other job J_i is set to $\max(T, t_{J_i})$, where t_{J_i} is the end of processing time of any operation from J_i being processed at time T in s.
4. The initial setup time of machine m is set to $\max(T, t_m) + \tau$, where t_m is the end of processing time of any operation being processed on m at time T in s. The initial setup time of any other machine M_j in the rescheduling problem is set to $\max(T, t_{M_j})$.

3 Minimax problems

The problem of finding flexible schedules is a *minimax problem*. A minimax problem can be formulated: minimise

$$\varphi(x) = \max_{s \in S} F(x, s) \quad \text{subject to } x \in X.$$

A minimax problem can be seen as an antagonist game between two players. The first player controls the variable x, often called *the solution*. The first players objective is to minimise $F(x, s)$. The second players objective is to maximise $F(x, s)$ by controlling s, often called *the scenario*. Often problems which can be seen as "the system against random attacks from nature" can be formulated as minimax problems.

Recently coevolution has been proposed to solve minimax problems [Bar97, Her99]. Coevolution seems ideally suited for solving minimax problems if both search-spaces (X and S) are large, prohibiting the use of exhaustive search when evaluating solutions. The idea when using coevolution to solve minimax problems is straight-forward: one population P_X represents the solutions $x_i \in X$, and another population P_S represents the scenarios $s_j \in S$. During fitness evaluation every solution in P_X is evaluated against every scenario in P_S. Individuals in P_X which do well against all individuals in P_S are assigned a high fitness, while individuals that perform poorly against some individual in P_S are assigned a low fitness. This is usually done by setting the fitness of each solution $x \in P_X$ to a decreasing function of $\max_{s \in P_S} F(x, s)$. In [Bar97, Her99] it is proposed to let the fitness of each scenario $s \in P_S$ be a decreasing function of $\min_{x \in P_X} F(x, s)$. In [Jen01b] it is demonstrated that this approach may be expected to work well for problems satisfying

$$\min_{x \in X} \max_{s \in S} F(x, s) = \max_{s \in S} \min_{x \in X} F(x, s), \qquad (1)$$

while poor performance should be expected if this condition is not satisfied. The problem is that calculating the fitness from $\min_{x \in P_X} F(x, s)$ favours scenarios that cause moderately high F values for all solutions, and avoids scenarios that cause low F values for some solutions, even if they cause very high F values for other solutions.

In [Jen01b] a more complex fitness evaluation for the scenarios is proposed to solve this problem. This fitness evaluation is based on the idea that a scenario

that causes a high $F(x, s)$ value relative to the other scenarios in P_S for at least one solution in P_X should be assigned a high fitness, while scenarios that do not cause relatively high F values for any solutions in P_X should be assigned a low fitness. This is done by evaluating $F(x_i, s_j)$ for each combination of solution and scenario $x_i \in P_X, s_j \in P_S$. For each x_i the scenarios are sorted into ascending order of $F(x_i, s)$. The fitness of each scenario $s \in P_S$ is set to the maximum index achieved by s in the orderings found. A small fitness contribution is added if a scenario gets its maximum index on several solutions. In [Jen01b] this fitness evaluation is demonstrated to work well for a few simple problems not satisfying (1).

4 The scheduling algorithms

Three different scheduling algorithms are used. All of the systems use a variant of the same genetic algorithm. Sometimes more traditional scheduling methods (e.g., the shifting bottleneck heuristic or branch and bound techniques) have shown performance superior to that of GAs, but the algorithms were based on GAs because GAs have previously demonstrated acceptable performance on scheduling problems, and because the coevolutionary idea is not compatible with traditional scheduling methods.

The first GA simply minimises the preschedule cost (makespan). This GA is referred to as the *preschedule performance GA*. It is used mostly to verify that the worst case performance after rescheduling is improved when using the other two algorithms. The second GA optimises the after breakdown and rescheduling performance of the schedules. The fitness evaluation is done in an exact way, making sure the worst case breakdown is tested by trying a large number of breakdowns. This algorithm, termed the *exact evaluation GA*, is very slow. The third GA also optimises after breakdown and rescheduling performance, but this is done by letting the schedule population coevolve with a population of breakdowns. In this way time can be saved when compared to the exact evaluation GA, at the expense of having some degree of noise in the fitness evaluation. The breakdown population size μ and number of progeny λ are important parameters in these algorithms, so they are termed *coevolutionary* $(\mu + \lambda)$ *algorithms*.

A very important decision in scheduling systems like these is how to do rescheduling. In a real world scheduling system after a breakdown has happened it would make sense to run the entire scheduling algorithm again, spending some time on finding a near optimal schedule. However, this is not possible when the preschedule has to be found, since rescheduling has to be performed a huge number of times during a single run of the algorithm. Rescheduling must be fast. One choice is to use right-shifting, but this is not a good choice when worst case performance is considered; whenever a breakdowns strikes at a *critical operation* (an operation that cannot be delayed without worsening the performance of the entire schedule) the makespan of the schedule will always be increased by the downtime. A possible solution to these problems is to use a local search technique for rescheduling: it is reasonably fast, and it can be able to improve on schedules that are broken in critical places. In all experiments in this paper the rescheduling is done by a hillclimber. The hillclimber identifies *critical blocks* (a critical block is a number of consecutive critical operations on the same machine) in the schedule and improves the schedule iteratively trying a number of permutations of each block. The reader is referred to [Mat96] for a detailed description of this hillclimber.

The following details hold for the scheduling part of all the genetic algorithms.

- The schedule representation used is called *permutation with repetition*. A schedule is represented by a sequence of job numbers, for instance the genome $(1, 2, 1, ...)$ can be decoded "first schedule the first operation of job 1, then the first operation of job 2, then the second operation of job 1, ...". Decoding the gene in this way creates a *semi-active* schedule; a schedule in which no operation can be scheduled earlier without changing the processing order. This representation has the advantage that no infeasible schedule can be represented. A number of other representations can be found in [Bru97].
- The schedule decoder is based on the same hillclimber used for rescheduling. An initial schedule is made using a semi-active schedule builder. This schedule is then improved by the hillclimber. The improved schedule is written back to the gene (Lamarckian learning).
- All new individuals were created using crossover. The crossover used is the Generalised Order Crossover (GOX). Each new individual was mutated with a probability of 0.1. The mutation operator is Position Based Crossover (PBM), see [Mat96].
- Tournament selection with a tournament size of two is used. The elite size is one.
- A population size of 100 is used, and the algorithms run for 100 generations.

In the preschedule performance GA, the objective function is the makespan of the preschedule. In the exact evaluation GA it is $\max_{s \in S} F(x, s)$. In the coevolutionary algorithm the objective function of the

schedules is $\max_{s \in P_S} F(x, s)$.

In the coevolutionary algorithm, the breakdown population evolves in a $(\mu+\lambda)$-evolutionary strategy. The population size is μ, and in each generation λ new individuals are generated. The new individuals compete with their parents and in every generation the λ worst individuals are discarded.

Each breakdown is represented by a breakdown time $t \in B_T$ and the machine breaking down $m \in B_M$. Remember that the downtime τ_B is fixed by the breakdown set. No crossover is used on the breakdowns; they only breed by mutation. Selection for breeding is done by linear ranking based selection.

When a breakdown is mutated, in 50% of the cases only the breakdown time is changed. The breakdown time is perturbed by adding a Gaussian distributed value with zero mean and standard deviation $\frac{1}{4}(T_{max} - T_{min})$. In 25% of the cases only the machine is changed, it is set to a random machine in B_M. In the last 25% of the cases, the individual is a completely random individual drawn uniformly from B.

The fitness evaluation used in the coevolutionary algorithm is the fitness evaluation presented in [Jen01b]. It is necessary to use a fitness evaluation of this kind, since the problem does not satisfy (1).

Due to the nature of job shop schedules, it is not necessary to consider all breakdowns in B in order to calculate the worst case performance of a schedule. Consider the breakdown during processing of the operation marked "X" on Figure 1. Any solution to the rescheduling problem for the breakdown time marked by "b" is bound to also be a solution to the breakdown with the breakdown time marked "a", since the breakdown marked "b" is more constrained than "a", while the processing of the of operation "X" will finish at the same time for both breakdown times (the preschedule finishing time plus the downtime). On the other hand, there exist solutions to the rescheduling problem of "a" that are not solutions to "b" (since the operation "Y" can be rescheduled for "a", but not for "b"). For these reasons the best possible solution to the rescheduling problem "b" will always be no better than the best possible solution to the rescheduling problem "a". Generally a breakdown can never be worsened by rounding up the breakdown time to the time just prior to the finishing time of an operation being processed at breakdown time.

Therefore when evaluating the worst case performance for a given schedule in the exact evaluation GA, only breakdown times that are immediately before the end of processing of an operation need to be considered,

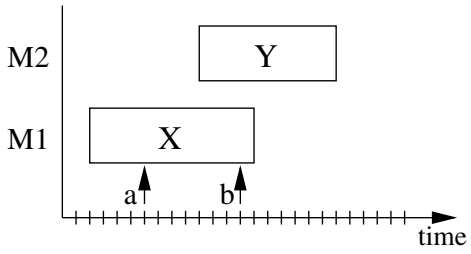

Figure 1: Rounding up a breakdown time.

along with breakdowns at time T_{max}. In the same way in the coevolutionary GA, when evaluating $F(x, s)$ for a given schedule $x \in P_X$ and breakdown $s \in P_S$, the breakdown time is rounded up to the time just before the end of processing of the current operation, or to T_{max}.

5 Experiments

The scheduling problem instances prefixed by `la` are from [Law84] and the problems prefixed by `ft` are from [FT63].

For each scheduling problem a breakdown set was created. This was done by inspecting a number of near-optimal schedules and selecting machines and times for each problem in such a way that parts of the schedules that would always be (near) critical were not included in the breakdown sets. The details of the breakdown sets and a little information on the scheduling problems can be seen in Table 1.

For each scheduling problem, six different sets of runs were performed. Four different runs with the coevolutionary $(\mu + \lambda)$-algorithms with $(\mu + \lambda)$ taking the values $(4 + 2), (8 + 4), (12 + 6)$ and $(16 + 8)$ to determine the effect of the breakdown population size. One run of the preschedule performance algorithm to determine the worst case breakdown performance improvement of the coevolutionary algorithms, and a run of the exact evaluation GA to determine the effect of the noise present in the fitness evaluation of the coevolutionary GA.

5.1 Results relevant to Scheduling

The average worst case makespan after a breakdown and rescheduling can be seen in Table 2. The averages have been calculated over 400 runs. It can be seen that for the two high values of $(\mu + \lambda)$ in all cases there is an improvement in the worst case performance. In some cases the improvement is substantial (problems `la01, la07, la31`), while in other cases it is modest

Problem	problem size	optimal preschedule makespan	breakdown times B_T	machines B_M	downtime τ_B	best found worst case makespan
la01	10×5	666	0-299	1,2,3,4	80	689
la02	10×5	655	0-299	1,2,3,5	80	713
la06	15×5	926	0-299	2,3,4,5	80	926
la07	15×5	890	0-299	2,3,4,5	80	890
la26	20×10	1218	0-599	2,3,4,6,7,8,9	80	1256
la27	20×10	1235	0-599	1,2,3,5,6,8,9,10	80	1290
la31	30×10	1784	0-599	2,3,4,5,6,8,9	80	1784
la36	15×15	1268	0-599	1,2,3,4,6,7,8,9,10	80	1316
ft10	10×10	930	0-299	3,4,5,6,7,8,9,10	80	989
ft20	20×5	1165	0-599	1,2	80	1190

Table 1: The problem sizes and breakdown sets used in the experiments. The optimal preschedule makespans can be found in [Mat96] and the references therein.

Problem	(4+2)	(8+4)	(12+6)	(16+8)	Presch. perf.	Exact eval.
la01	692.0	689.5	689.1	689.2	725.7	689.0
la02	741.8	735.8	733.4	732.5	735.8	730.8
la06	926.2	926.1	926.0	926.0	949.4	926.0
la07	897.2	892.9	892.1	892.0	945.6	892.1
la26	1288.1	1282.5	1279.7	1279.2	1293.5	1277.2
la27	1343.2	1336.8	1333.4	1330.8	1343.5	1328.5
la31	1801.2	1794.3	1792.6	1788.6	1830.9	1784.0
la36	1358.3	1347.1	1341.1	1339.7	1364.6	1337.5
ft10	1020.4	1017.1	1018.1	1017.5	1037.4	1018.3
ft20	1254.5	1250.5	1249.9	1248.2	1266.2	1247.5
Average	1132.3	1127.3	1125.5	1124.4	1149.3	1123.0

Table 2: Average worst case performances.

(la02). For the two small values of $(\mu+\lambda)$, the performance is generally worse than the performance for the high values. In one case (la02) there is even a drop in the worst case performance when compared to the preschedule performance GA.

Considering the makespan performance of the preschedules without breakdown and rescheduling (Table 3), it is evident that for some of the problems the increased flexibility observed in Table 2 comes at a cost in preschedule performance. For la02, la27, la36, ft10 and ft20, the preschedule makespan is increased by 10 or more by using the $(16+8)$ coevolutionary GA instead of the preschedule performance GA. In other cases, la06, la07 and la31 there is no increase in preschedule makespan at all.

The variation in after breakdown performance and preschedule performance from problem to problem indicates that for some problems and breakdown sets optimising worst case performance after breakdowns using a coevolutionary GA seems to perform very well. Consider la07 and la31, where a substantial improvement in worst case performance comes at no cost in preschedule performance. For other problems the performance is quite poor. For la02 and la27 a small or modest performance increase after rescheduling comes at a high price in preschedule performance. These observations indicate that if a scheduling system like the coevolutionary GA is to be used in the real world, great care will have to be taken. A way of circumventing this problem could be to create a multi-objective version of the algorithm, that would optimise preschedule performance as well as worst case performance. The system would return a Pareto front of non-dominated solutions, and a human expert would decide which schedule to implement.

5.2 Results relevant to Evolutionary Computation

From Table 2 it is evident that the noise present in the fitness evaluation of the coevolutionary GA can have a negative effect on performance. For the small values of $(\mu+\lambda)$, in most cases the performance is a bit worse than the performance of the exact evaluation GA. For the higher values of $(\mu+\lambda)$, the fitness evaluation noise is smaller due to better sampling of the breakdown search-space, and the performance seems to be almost as good as that of the exact evaluation GA.

The effect of the population size μ on the noise in the fitness evaluation of the final individual has been investigated for the la07 problem in the left plot of Figure 2. On the plot it is evident that there is a significant drop in noise when increasing μ from 4 to 8, while smaller drops arise when increasing μ to 12 and 16. The effect of the population size on worst case performance and the CPU time spent has been visualised on the middle and right plots of Figure 2,

The average processing times for one run of each al-

Problem	(4+2)	(8+4)	(12+6)	(16+8)	Presch. perf.	Exact eval.
la01	666.8	666.7	666.8	667.0	666.0	667.1
la02	674.7	669.8	668.0	666.2	657.7	665.6
la06	926.2	926.1	926.0	926.0	926.0	926.0
la07	890.0	890.1	890.0	890.0	890.0	890.0
la26	1225.2	1224.3	1223.1	1223.3	1218.5	1222.8
la27	1285.5	1284.0	1283.3	1282.1	1267.4	1282.3
la31	1784.1	1784.0	1784.0	1784.0	1784.0	1784.0
la36	1315.7	1316.3	1315.8	1315.9	1297.3	1316.1
ft10	986.2	985.0	986.3	986.9	959.6	986.6
ft20	1206.7	1204.5	1205.6	1204.1	1192.7	1205.3
Average	1096.1	1095.1	1094.9	1094.6	1085.9	1094.6

Table 3: Average preschedule performance without breakdown.

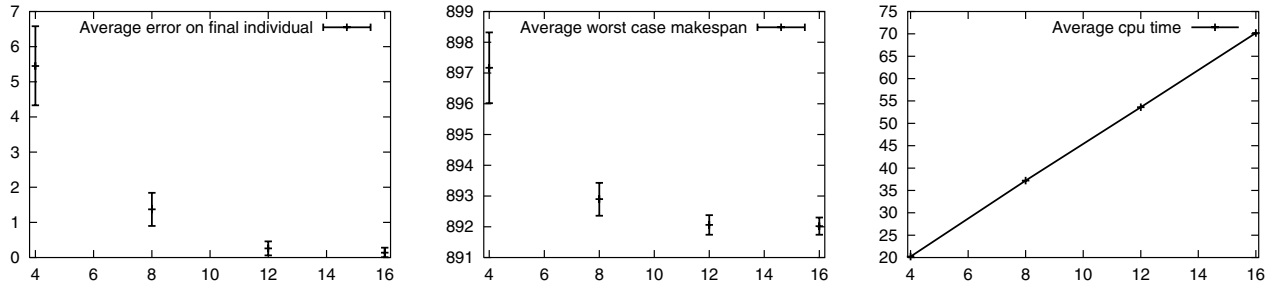

Figure 2: Plots for the coevolutionary runs on the la07 problem. **Left:** The average error on the fitness evaluation of the final individual for various values of μ ($\lambda = \frac{1}{2}\mu$). The error bars indicate 95% confidence intervals on the average. **Middle:** Average worst case makespan. **Right:** CPU time used in seconds.

gorithm can be seen on Table 4. The experiments were run on a 250MHz SGI O2 computer. It is evident that even for small values of $(\mu + \lambda)$ the coevolutionary GAs are much slower than the preschedule performance GA. This is due to the processing time spent doing rescheduling and evolving the breakdown population. Generally, the processing time of the coevolutionary GA seems to increase linearly with the breakdown population size μ.

Comparing the processing times of the exact evaluation GA and the coevolutionary GA, it seems that for the smaller problems (la01, la02, la06, la07, ft10 and ft20) only a modest amount of processing time is saved, and only if a small breakdown population size μ is used. For breakdown population sizes of $\mu = 16$ in some cases the coevolutionary GA is slower than the exact evaluation GA. For the small values of μ some processing time is saved (typically 50%-70% for $\mu = 4$ and 7%-50% for $\mu = 8$). Given the slightly superior quality of the solutions found by the exact evaluation GA it seems that for small problems the exact evaluation GA should be preferred unless time is very critical.

For the larger problems la26, la27, la31 and la36, more time can be saved. For the smallest breakdown population size $\mu = 4$ around 90% of the processing time is saved. For the largest breakdown population size $\mu = 16$, approximately 65% of the processing time is saved. These are substantial savings, since the processing time for a run of the exact evaluation GA is more than 10 CPU minutes for all of these problems. For bigger problems the time saved is expected to be bigger. Which population size to use is a tradeoff, since the quality of schedules produced is better the bigger μ is.

6 Conclusion

A minimax formulation of job shop scheduling to achieve the best possible worst case performance given a set of possible breakdowns has been presented. This problem has been solved using three different genetic algorithms. One minimises the preschedule cost. One minimises the worst case cost after a breakdown and rescheduling by optimising an exact measure of worst case cost. One minimises the worst case cost after a breakdown and rescheduling by estimating the worst case performance of the schedules using a population of breakdowns that coevolves with the schedule population.

It has been demonstrated that the worst case performance of the schedules can indeed be improved by using the coevolutionary GA or the GA optimising the worst case performance. How big this improve-

Problem	(4+2)	(8+4)	(12+6)	(16+8)	Presch perf.	Exact eval.
la01	13.2	28.3	35.8	47.4	1.5	46.9
la02	14.9	27.4	38.2	50.7	1.3	49.5
la06	20.3	37.2	54.1	70.9	3.2	65.4
la07	20.2	37.4	53.6	70.2	3.3	73.8
la26	70.3	128.3	186.1	244.1	13.6	669.2
la27	78.4	138.7	202.9	257.1	13.9	734.6
la31	154.8	270.2	392.5	505.0	31.1	1375.1
la36	71.5	131.0	184.8	247.0	10.8	824.6
ft10	31.2	60.7	88.4	117.9	3.5	65.0
ft20	39.9	73.9	109.1	137.2	5.8	141.8
Average	51.5	93.3	134.6	174.8	8.8	404.6

Table 4: Average processing time in CPU-seconds.

ment is depends on the scheduling problem and the set of breakdowns. For some problems the improvement comes at the cost of decreasing preschedule performance when no breakdown occurs.

The experiments have shown that for large problem instances the coevolutionary algorithm clearly outperforms the exact evaluation algorithm in terms of processing speed, while finding schedules of a slightly lower quality. For the coevolutionary algorithms a tradeoff has been demonstrated; for small breakdown population sizes the processing is fast. For larger breakdown population sizes the processing is slower, while the schedule quality increases.

Future work includes experiments with a larger set of problem instances, as well as larger sets of breakdowns. Changing the algorithm to work with deviation worst case performance also seems an interesting line of research.

Because of the tradeoff between preschedule cost and worst case performance, a very interesting line of research would be to make a multi-objective optimisation algorithm that optimised preschedule performance and worst case performance at the same time, and returned a set of Pareto-optimal solutions. The solution to be implemented should then be hand-picked by a human expert.

References

[Bar97] H. J. C. Barbosa. A coevolutionary genetic algorithm for a game approach to structural optimization. In *Proceedings of the seventh International Conference on Genetic Algorithms*, pages 545–552, 1997.

[BM00] J. Branke and D. C. Mattfeld. Anticipation in dynamic optimization: The scheduling case. In M. Schoenauer et al., editor, *Parallel Problem Solving from Nature - PPSN VI proceedings*, LNCS vol. 1917, pages 253–262. Springer, 2000.

[Bru97] R. Bruns. Scheduling. In T. Bäck et. al., editor, *Handbook of Evolutionary Computation*, chapter C1.5. IOP Publishing and Oxford University Press, 1997.

[FT63] H. Fisher and G.L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J.F. Muth and G.L. Thompson, editors, *Industrial Scheduling*, pages 225–251. Prentice Hall, 1963.

[Her99] J. W. Herrmann. A genetic algorithm for minimax optimization problems. In P. J. Angeline et. al., editor, *Proceedings of the 1999 Congress on Evolutionary Computation*, volume 2, pages 1099–1103. IEEE Press, 1999.

[Jen01a] M. T. Jensen. Improving robustness and flexibility of tardiness and total flowtime job shops using robustness measures. *Journal of Applied Soft Computing*, 2001. To appear.

[Jen01b] M. T. Jensen. A remark on solving minimax problems with coevolution. In *Proceedings of GECCO 2001*, 2001.

[KY97] P. Kouvelis and G. Yu. *Robust Discrete Optimization and Its Applications*. Kluwer Academic Publishers, 1997.

[Law84] S. Lawrence. *Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement)*. Graduate School of Industrial Administration, Carnegie-Mellon University, 1984.

[LWS94] V. J. Leon, S. D. Wu, and R. H. Storer. Robustness measures and robust scheduling for job shops. *IIE Transactions*, 26(5):32–43, September 1994.

[Mat96] D. C. Mattfeld. *Evolutionary Search and the Job Shop*. Production and Logistics. Physica-Verlag, 1996.

A Fuzzy Theory Based Evolutionary Approach for Driver Scheduling

Jingpeng Li
School of Computing
University of Leeds
Leeds, United Kingdom
li@comp.leeds.ac.uk

Raymond S.K. Kwan
School of Computing
University of Leeds
Leeds, United Kingdom
rsk@comp.leeds.ac.uk

Abstract

Based on fuzzy set theory, this paper presents a novel evolutionary approach for the driver scheduling problem, which involves solving a set covering model. At the heart of this approach is a function for evaluating, under fuzzified criteria, potential driver shifts. A Genetic Algorithm is first employed to calibrate the weight distribution among fuzzy membership functions. A Simulated Evolution algorithm then mimics generations of evolution on a single schedule. In each generation an unfit portion of the working schedule is removed. The broken schedule is then reconstructed by means of a greedy algorithm. Experiments using data from the transportation industry have demonstrated that the evolutionary approach is suitable for large size driver scheduling problems. It is suggested that this approach might be applied to other large-scale set covering problems.

1 INTRODUCTION

Bus and rail driver scheduling is a process of partitioning blocks of work, each of which is serviced by one vehicle, into a set of legal driver shifts. At the operational planning stage, the driver shifts are only notional, i.e. they are not compiled with specific drivers in mind. The basic objectives are to minimize the total number of shifts and the total shift costs. If the scheduling process is integrated with the assignment of actual personnel, a more complicated multi-criterion model may be appropriate (e.g. Cai and Li (2000)). This problem has been the subject of research since the 1960's. Wren and Rousseau (1995) gave an overview of the approaches, many of which have been reported in a series of international workshop conferences (Desrochers and Rousseau, 1992; Daduna et al., 1995; Wilson, 1999).

In driver scheduling, a *Relief Opportunity* (RO) is a time and place where a driver can leave the current vehicle, for reasons such as taking a meal-break, or transferring to another vehicle. The work between two consecutive ROs on the same vehicle is called *a piece of work*. The work a single driver carried out in a day is called *a shift*, which is composed of several *spells* of work. A *spell* contains a number of consecutive pieces of work on the same vehicle, and a *schedule* is a solution that contains a set of shifts that cover all the required driver work.

Driver scheduling can be modeled as a set covering problem, which is NP-hard (Chvátal, 1979). Possible legal shifts, usually a very large set, are first generated by specific heuristics. Then, a least cost subset covering all the work is selected to form a solution schedule. For example, the TRACS II system (Proll, 1997; Fores et al., 1999) is well-known and is based on the set covering model. TRACS II uses a blend of heuristics and Integer Linear Programming (ILP), the success and limitations of which have been discussed in Kwan et al. (2000).

Since the set covering problem is unlikely to be solved optimally in polynomial time, a lot of work has been undertaken to explore the possibility of obtaining efficiently near-optimal solutions. One of these polynomial time algorithms is the greedy algorithm: at each step choose the unused set which covers the largest number of remaining elements. However, the simple greedy method is not suitable for large size set covering problems due to its poor approximation guarantee (Lovász, 1975). In this paper, we present a fuzzy theory based evolutionary approach.

First, a function for evaluating the potential shifts is designed. Criteria used are characterized by fuzzy membership functions, which would lead to a quantitative formulation of the structural goodness of a shift by a simplified method of fuzzy comprehensive evaluation. A Genetic Algorithm (GA) is applied to calibrate the weight distribution among fuzzy membership functions.

Secondly, a Simulated Evolution (SE) algorithm combines the features of iterative improvement and constructive perturbation with the ability to avoid getting stuck at poor local minima. Two main steps, *Evaluation* and *Reconstruction*, have been fuzzified: In the *Evaluation* step, each shift in the current solution is evaluated by the above evaluation function. In the

Reconstruction step, a greed-based heuristic using this fuzzy evaluation function is applied to repair the broken schedule.

This paper is organized as follows. Section 2 introduces the method of fuzzy evaluation of shift fitness. Section 3 discusses the SE algorithm. The determination of weights by a GA is presented in section 4. Benchmark results using real-world problems are given in section 5. Concluding remarks are in section 6.

2 FUZZY COMPREHENSIVE EVALUATION

From the viewpoint of driver scheduling, the vehicle schedule consists of a set of pieces of work I={1, 2, …, m} to be covered. A very large set of potential shifts S={S_1, S_2, …, S_n} has been generated. Each shift covers a subset of the pieces of work ($S_j \subseteq I$ for $j \in J$={1, 2, …, n}), and has an associated cost c_j (hours paid). A subset of shifts $(J^* : J^* \subseteq J)$ covers all the work if $\bigcup (S_{j^*} : j^* \in J^*) = I$.

The process of constructing a potential schedule by the greedy heuristic is inherently sequential. However, among the large set of potential shifts, it would be difficult to judge which one is more effective than others because the criteria bear some uncertainty. To mitigate the problem, fuzzy comprehensive evaluation, a powerful tool to describe quantitative uncertain values and relations between them, is used to introduce the concept of structural coefficient. It gives shift $S_j (j \in J)$ a quantitative value $f_1(S_j) \in [0,1]$ according to its structural state. The fitter the structure for S_j, the larger $f_1(S_j)$ is.

The main idea of this principle is to set up several criteria characterized by fuzzy membership functions. Unlike traditional criterion, fuzzy comprehensive evaluation can make decision based on all of the fuzzified criteria. Considering the structural state of a shift in more aspects, the result will be more reliable than conventional approaches in deciding the efficiency of the shift.

There are two steps in establishing the new principle. First, a number of fuzzified criteria should be obtained according to the efficiency of a shift, which describes quantitatively the characteristic of its structural state from different aspects. Secondly, fuzzy comprehensive evaluation will be applied to appraise effectively the shift structural state for decision making. These two steps are presented respectively as follows.

2.1 CONSTRUCTION OF THE FUZZIFIED CRITERIA

As far as the shift structure is concerned, the main criteria are total worked time u_1, ratio u_2 of total worked time to spreadover (normally the paid hours for a driver form sign on to sign off), number of pieces of work u_3, and number of spells u_4 contained in a shift.

A common method for shift selection is Integer Linear Programming, which is NP-hard (Garey and Johnson, 1979). Large problems would have to be divided into sub-problems, and in some cases the ILP process may have difficulties in finding an integer solution. In contrast, the relaxed problem in which the solution vector is not required to be integral, $X \in [0,1]^n$, is much easier: the optimal solution for the relaxed problem can be found in polynomial time (Karmarkar, 1984). Although Slavík (1996) proved the performance guarantee for the randomized rounding technique on fractional cover was even worse than that of the simple greedy algorithm, we still can assume that at least the relaxed solution provides some useful information about the optimum integer solution. Therefore, the fractional cover by Linear Programming (LP) relaxation u_5, if applicable, can be considered as another criterion.

2.1.1 Criteria u_1, u_2, and u_3

The goodness of a potential shift $S_j (j \in J)$ generally increases with its total worked time, ratio of total worked time to spreadover, and number of pieces of work. A similar formulation of the membership function μ_i (i=1, 2, 3) for these three factors is therefore used and defined as:

$$\mu_i = \begin{cases} 2\left(\dfrac{x_i - a_{\min}^{(i)}}{a_{\max}^{(i)} - a_{\min}^{(i)}}\right)^2, & a_{\min}^{(i)} \le x_i < \dfrac{a_{\min}^{(i)} + a_{\max}^{(i)}}{2} \\ 1 - 2\left(\dfrac{x_i - a_{\max}^{(i)}}{a_{\max}^{(i)} - a_{\min}^{(i)}}\right)^2, & \dfrac{a_{\min}^{(i)} + a_{\max}^{(i)}}{2} \le x_i \le a_{\max}^{(i)} \end{cases} \quad (1)$$

where x_1 = total worked time of S_j;
$a_{\max}^{(1)}$ = maximum total worked time;
$a_{\min}^{(1)}$ = minimum total worked time;
x_2 = ratio of total worked time to spreadover for S_j;
$a_{\max}^{(2)}$ = maximum ratio;
$a_{\min}^{(2)}$ = minimum ratio.
x_3 = number of pieces of work contained in S_j;
$a_{\max}^{(3)}$ = maximum number of pieces of work;
$a_{\min}^{(3)}$ = minimum number of pieces of work.

2.1.2 Criterion u_4

In most practical bus and rail scheduling problems, the number of spells in a potential shift is limited to be at most four. 2-spell shifts are generally more effective than others, and 3-spell shifts are more desirable than 1-spell or 4-spell shifts. Hence membership function μ_4 for the spell factor is defined as:

$$\mu_4 = \begin{cases} 0, & \text{if } x_4 = 1 \text{ or } x_4 = 4 \\ 1/2, & \text{if } x_4 = 3 \\ 1, & \text{if } x_4 = 2 \end{cases} \quad (2)$$

where x_4 = number of spells contained in S_j.

2.1.3 Criterion u_5

The fractional cover by LP relaxation provides some useful information about the significance of some of the shifts identified in the relaxed solution. According to experimentation in Kwan et al. (2000), the higher the fractional value of the variable for a shift, the higher chance that it is present in the integer solution. Hence membership function μ_5 for the fractional cover factor is defined as follows:

$$\mu_5 = \begin{cases} e^{-\frac{(x_5-\alpha)^2}{\beta}}, & \text{if } S_j \text{ is in the fractional cover} \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

Let $\mu_5 = 1$ when $x_5 = a_{\max}^{(5)}$, and $\mu_5 = 0.01$ when $x_5 = a_{\min}^{(5)}$, where x_5 = fractional value of S_j in the relaxed LP solution;

$a_{\max}^{(5)}$ = maximum value in fractional cover;

$a_{\min}^{(5)}$ = minimum value in fractional cover.

Therefore,

$$\begin{cases} \alpha = a_{\max}^{(5)} \\ \beta = -\dfrac{(a_{\max}^{(5)} - a_{\min}^{(5)})^2}{\ln 0.01} \end{cases} \quad (4)$$

2.2 FUZZY COMPREHENSIVE EVALUATION

Fuzzy set theory is a means of presenting uncertainty put forward by Zadeh (1965). It has been developed to improve the oversimplified model and makes more flexible and robust models to solve real-world complex problems (Dubois and Prade, 1980; Klir and Yuan, 1995).

Based on rationale of fuzzy mathematics, fuzzy comprehensive evaluation considers various criteria affecting the structure of a shift in a mathematical model to evaluate the efficiency of a shift.

Therefore, for shift $S_j (j \in J)$, the formulation of its structural coefficient $f_1(S_j)$ by the method of fuzzy comprehensive evaluation is:

$$f_1(S_j) = \sum_{i=1}^{5} (w_i \times \mu_i), \forall j \in J \quad (5)$$

Where, w_i denotes the corresponding weights for criteria u_i (i=1, 2, 3, 4, 5). They all satisfy the normalizing condition $\sum_{i=1}^{5} w_i = 1$ and $w_i \geq 0$. If the i-th criterion were dominant in assessing the shift structure, its weight should have a high value.

From the analysis above it can be seen that the main task of fuzzy comprehensive evaluation for structural coefficient of a shift is to determine the weight distribution among the fuzzy membership functions. Genetic Algorithms rather than experience could be applied to determine the weights, which is described in section 4.

3 A FUZZY SIMULATED EVOLUTION ALGORITHM

Simulated Evolution (SE) algorithm is a general optimization technique originally proposed by Kling and Banerjee (1987) for the Placement problem and subsequently applied by other researchers to optimization problems in the CAD area (Lin et al., 1989; Ly and Mowchenko, 1993; Sait et al., 1999).

In this section a fuzzy SE algorithm is described. It iteratively operates a sequence of *Evaluation*, *Selection, Mutation* and *Reconstruction* steps on a single schedule. Besides these three steps, some input parameters, e.g. stopping condition, and a valid starting solution are initialized in an earlier step called *Initialization*. Throughout these iterations, the best solution is retained and finally returned as the final solution. This algorithm is a greedy search strategy that achieves improvement through iterative perturbation and reconstruction.

The SE requires as input a set of legal potential shifts to select from. The heuristics for generating such a set of legal potential shifts is complex, taking into account many labor agreement rules and constraints, and are not the subject of this paper. Here we make direct use of the TRACS II system to provide the set of potential shifts.

3.1 INITIALIZATION

In this step, an initial solution is generated to serve as a seed for the evolutionary process. Explained in Section 4, the GA for calibrating the weight distribution of the fuzzy evaluation function would provide, as a by-product, a good initial solution for the SE. The steps described in section 3.2 to 3.5 are executed in sequence in a loop until a user specified parameter (e.g. cpu-time, total cost, or number of shifts) is reached or no improvement has been achieved for a certain number of iterations.

3.2 EVALUATION

In this step, goodness of the individual shift in a complete schedule J^* is computed. The evaluation function $F(S_{j^*})$ for shift $S_{j^*} (j^* \in J^*)$ should be normalized. Besides the structural coefficient $f_1(S_{j^*})$, another normalized function, which reflects the coverage status for shift S_{j^*}, should be combined. Hence our evaluation function $F(S_{j^*})$ consists of two parts: structural coefficient $f_1(S_{j^*}) \in [0,1]$ and over-cover penalty $f_2(S_{j^*}) \in [0,1]$, which can be formulated as:

$$F(S_{j^*}) = f_1(S_{j^*}) \times f_2(S_{j^*}), \qquad \forall j^* \in J^* \quad (6)$$

The ratio of the overlapped work time to total work time in $S_{j^*} (j^* \in J^*)$ is also regarded as an important criterion, which can be formulated as over-cover penalty $f_2(S_{j^*}) \in [0,1]$ below.

$$f_2(S_{j^*}) = \sum_{k=1}^{|S_{j^*}|} (\alpha_{j^*k} \times \beta_{j^*k}) \Big/ \sum_{k=1}^{|S_{j^*}|} \beta_{j^*k}, \quad \forall j^* \in J^* \qquad (7)$$

where $|S_{j^*}| =$ number of pieces of work in S_{j^*};

$$\alpha_{j^*k} = \begin{cases} 0 & \text{if work piece } k \text{ in } S_{j^*} \text{ has been covered} \\ & \text{by any other shift } S_i \text{ in } J^*; \\ 1 & \text{otherwise;} \end{cases}$$

$\beta_{j^*k} =$ worked time for work pieces k in S_{j^*}.

If every piece of work in S_{j^*} has been covered by other shift S_i in J^*, then $f_2(S_{j^*}) = 0$; conversely if none of the pieces of work in S_{j^*} is overlapped, $f_2(S_{j^*}) = 1$.

3.3 SELECTION

In this step it will be determined whether a shift $S_{j^*}(j^* \in J^*)$ is retained for the next generation, or discarded and placed in a queue for the new allocation. This is done by comparing its goodness $F(S_{j^*})$ to $(p_s - k)$, where p_s is a random number generated for each generation in the range [0, 1], and k is a constant smaller than 1.0. If $F(S_{j^*}) > (p_s - k)$ then S_{j^*} will survive in its present position; otherwise S_{j^*} will be removed from the current evolutionary schedule. The pieces of work it covers, except those also covered by other shifts in the solution, are then released for the next *Reconstruction*. By using this *Selection* process, shift S_{j^*} with larger goodness $F(S_{j^*})$ has higher probability of survival in the current schedule.

The purpose of subtracting k from p_s is to improve the SE's convergent capability. Without it, in the case of p_s close to 1, nearly all the shifts will be removed from the schedule, which is obviously ineffective in searching. In our experiments, we set k to be 0.3.

3.4 MUTATION

To escape from local minima in the solution space, capabilities for uphill moves must be incorporated. This is carried out in the *Mutation* step by probabilistically discarding even some superior components of the solution. Therefore, following the *Selection* step, each retained shift $S_{j^*}(j^* \in J^*)$ has a chance to be mutated, i.e. randomly discarded from the partial solution at a given rate of p_m, and releases its covered pieces of work, except those also covered by other retained shifts, for the next *Reconstruction*. The mutation rate should be much less than the selection rate to guarantee convergence. From empirical results we find that $p_m \le 5.0\%$ yields better results.

3.5 RECONSTRUCTION

The *Reconstruction* step takes a partial schedule as the input, and produces a complete schedule as the output. Since the new schedule should be an evolution of the previous schedule, all shift assignments in the partial schedule should remain unchanged. Therefore, the *Reconstruction* task reduces to that of assigning shifts to all uncovered pieces of work to complete a partial schedule.

Each of the remaining unassigned work pieces i has a coverage list of length L_i, i.e. containing L_i potential shifts that can cover it. The greed-based constructor assumes that the desirability of adding shift $S_j(j \in J)$ into the partial schedule increases with its goodness value $F(S_j)$. However, to introduce diversification, we randomly select one of the candidates, not necessarily the top candidate, from a Restricted Candidate List (RCL), which consists of k best shifts. From empirical results we find that $k \le 4$ achieves better solutions. The steps to generate a complete schedule are:

Step 1 $J^* = \{1,2,...,l\}$ is a partial schedule.

Step 2 Set $I' = I - \bigcup (S_{j^*} : j^* \in J^*)$.

Step 3 If $I' = \Phi$ then stop: J^* is a complete schedule and $C(J^*) = \sum (c_{j^*} : j^* \in J^*)$. Otherwise randomly select a shift S_k within RCL from the shortest coverage list and proceed to step 4.

Step 4 Add k to J^*, set $I' = I' - S_k$, and return to step 3.

It should be noted that the shifts added during schedule *Reconstruction* might be redundant, causing all their pieces of work covered by other shifts later, even if each shift is chosen to cover at least one currently uncovered piece of work. However, in the next *Selection*, these redundant shifts will be removed automatically because of their zero goodness. Moreover, the goodness values of all shifts in the current *Construction* might be different from those in the next *Selection* as well due to the updated over-cover penalties at each iteration.

4 A GENETIC ALGORITHM TO DETERMINE WEIGHTS

The designed evaluation function, parameterised by the weights of the five fuzzy membership functions, plays a key role in the SE algorithm: it directs the *Selection* and *Reconstruction* steps.

Based on the mechanics of genetics and natural selection, GAs (Goldberg, 1989) are useful approaches to problems requiring an efficient search in a very large solution space, and can be used to obtain approximate solutions for multivariable optimization problems. Therefore, a GA is proposed to determine the weight distribution. Since the fitness of a set of weights is assessed by applying it to construct a schedule, as a by-product of this process, the schedule associated with best set of weights is used as the initial solution for the SE.

Similar to the *Reconstruction* step of SE, a complete schedule is obtained iteratively from $J^* = \Phi$. The weight distribution for $F(S_j)$ is evolved by a GA outlined as follows:

Step 1 Set generation $t = 0$; initial population $P(t)$ is generated with randomised weights.
Step 2 Apply the *Reconstruction* step above and evaluate the members in $P(t)$.
Step 3 If termination criterion is reached then stop; otherwise proceed to step 4.
Step 4 Set $t = t + 1$; select members from $P(t\text{-}1)$ for reproduction.
Step 5 Perform crossover and mutation to produce offspring, and partially replenish $P(t)$ by randomly generated members. Apply the above *Reconstruction* and evaluate all new members and return to step 3.

4.1 CHROMOSOME REPRESENTATION

The first step is to represent the weights in a way suitable for applying the genetic operators. The weights w_i (i=1, 2, 3, 4, 5; $w_i \in [0,1]$) are continuous variables requiring an integer representation. Each variable is first linearly mapped to an integer defined in a specified range, and the integer is encoded using a fixed number of binary bits. The binary codes of all the variables are then concatenated to obtain a binary string.

In our experiments, w_i is encoded in 6 binary bits. Hence, the problem is a 5-dimension-search, and the solution space is 2^{30}.

4.2 MEASUREMENT OF FITNESS

The fitness function can be designed as the total cost of the shifts in the schedule, plus a sufficiently large multiple of the number of shifts to ensure that priority is given to minimizing shift numbers. The lower the weighted cost of the schedule, the fitter the chromosome is.

4.3 SELECTION

Selection models nature's survival-of-the-fittest mechanism. Fitter solutions survive while weaker ones perish. Here we use the traditional roulette wheel strategy. Member with the least cost in each generation are preserved if they have not been selected.

4.4 ADAPTIVE PROBABILITIES OF CROSSOVER AND MUTATION

There are two essential characteristics in GAs for optimising multi-modal functions. The first is the capacity to converge to a local or global optimum after locating the region containing the optimum. The second is the capacity to explore new regions of the solution space in search of the global optimum. The balance between these two characteristics is decided by values of Crossover Probability p_c and Mutation Probability p_m, and the type of crossover applied. Increasing values of p_c and p_m promotes exploration at the expense of exploitation.

To accomplish this trade-off between exploration and exploitation in a different manner, Srinivas (1994) designed an algorithm that could vary p_c and p_m adaptively in response to the fitness values of the solutions: p_c and p_m are increased when the population tends to get stuck at a local optimum and are decreased when the population is scattered in the solution space.

Here we apply Srinivas's algorithm, formulated as below, and perform 5-point crossover and mutation operators to the five weights:

$$\begin{cases} p_c = k_1(f_{\max} - f')/(f_{\max} - \bar{f}), & f' \geq \bar{f} \\ p_c = k_2, & f' < \bar{f} \end{cases} \tag{8}$$

and

$$\begin{cases} p_m = k_3(f_{\max} - f)/(f_{\max} - \bar{f}), & f \geq \bar{f} \\ p_m = k_4, & f < \bar{f} \end{cases} \tag{9}$$

Where k_1, k_2, k_3, and k_4 are constants smaller than 1.0; f_{max} denotes the smallest cost value of the population; f' denotes the smaller cost value of the solutions to be crossed; $\bar{f}$ denotes the average cost value of the population; and f denotes the cost value of the solution to be mutated.

Based on the establishment that moderately large values of p_c ($0.5 < p_c < 1.0$) and small values of p_m ($0.001 < p_m < 0.05$) are important for the successful working of GAs (Goldberg, 1989), we set k_1, k_2 , k_3 , and k_4 to be 0.96, 0.96, 0.12, and 0.16 respectively to perform our experiments.

5 COMPUTATIONAL RESULTS

The two main objectives of minimizing cost and minimizing total number of shifts in a schedule are combined as a weighted sum cost function, i.e. minimizing $\sum_{j^*=1}^{l} c_{j^*} + l \times 2000$, where l is number of shifts in the schedule and c_{j^*} is the cost of shift S_{j^*}. In most driver scheduling problems the first objective is to minimize the number of shifts, and a large constant of 2000 per shift is used to give priority to this.

Table 1 shows the sizes and the best known results of eleven test problems, all of which are real world driver scheduling problems from medium to very large sizes (Kwan, 1999). The best known schedules are mostly obtained by the TRACS II system, which is a commercial system based on ILP with more than 100 person-years devoted in its development. In two cases where TRACS II has difficulty in finding solutions, results reached by hybrid Genetic Algorithms incorporating strong domain knowledge (Kwan et al., 1999; Kwan et al., 2000) are cited.

Table 1: Size and the Best Known Schedules of Test Problems

Data	Type	Number of pieces of work	Number of potential shifts	Best known schedule		
				Shifts	Cost (hours paid)	Elapsed time (secs.)
Colx	Bus	127	3560	34	288.16	22
Gmb	Bus	154	11817	34	289.32	84
Neur	Train	340	29380	62*	509.25*	955
Ews	Train	437	25099	116	1003.55	69
Wag3	Train	456	16636	50	403.42	34
Tram	Tram	553	6437	49	419.50	24
Trmx	Tram	553	29500	49	408.47	139
Nb2	Bus	613	22568	75*	851.09*	452
G532	Train	1164	29465	276	2083.15	>80000
Gall	Train	1495	28639	349	2661.12	>80000
Rrne	Train	1873	50000	395	3137.20	>40000

* Results of Nb2 and Neur cases are obtained by the hybrid GA, while others are obtained by the TRACS II system.

Table 2: Results of the New Evolutionary Approach
(Percentages are relative deviations relating to the best known solutions)

Data	Initial schedule derived by GA				SE's final schedule				
	Shifts	%	Cost (h)	%	Shifts	%	Cost (h)	%	Time (s)
Colx	36	5.88	302.51	4.98	35	2.94	294.06	2.05	24
Gmb	37	8.82	307.33	6.22	35	2.94	294.92	1.94	16
Neur	66	6.45	531.02	4.27	62	0.00	507.67	-0.31	120
Ews	118	1.72	1022.08	1.85	117	0.82	1000.18	-0.34	167
Wag3	51	2.00	416.65	3.28	51	2.00	406.55	0.78	11
Tram	51	4.08	442.10	5.39	49	0.00	421.56	0.49	23
Trmx	51	4.08	427.70	4.71	49	0.00	414.38	1.45	59
Nb2	76	1.33	881.92	3.62	74	-1.33	830.60	-2.41	216
G532	277	0.36	2152.38	3.32	271	-1.81	2104.33	1.02	130
Gall	350	0.29	2749.32	3.31	343	-1.72	2663.05	0.07	358
Rrne	407	3.04	3399.62	8.36	390	-1.27	3242.75	3.36	1320
Avg.		3.46%		4.48%		0.24%		0.74%	

In some cases, the ILP process of TRACS II fails to find an integer solution after a large number of nodes of the branch-and-bound search tree has been explored. In these circumstances, the target is raised by one shift and the ILP is re-run. The process is repeated if an integer solution still cannot be found, and maybe abandoned after the target has been raised many times without success (e.g. Neur and Nb2 instances).

The above evolutionary approach was coded in Borland C++. All problems were run on the same Pentium II 333 MHz with 196 megabyte RAM personal computer using Windows 98 operating system. If no improvement has been achieved for 1000 iterations, the program will terminate. Further more, we set p_m in *Mutation* of SE to be 5.0%, and size k of RCL in *Reconstruction* to be 3, and the population size of GA to be 100 to all problems. The benchmark experimental results in terms of shift number and total cost for the initial solutions (as a by-product of the weight distribution calibration GA) and the final SE solutions are compiled in table 2. Elapsed time is the time following the solution of the relaxed LP of TRACS II.

The new approach has successfully solved two problems which were not solved by TRACS II with better solutions and much faster speed than other heuristics, and has produced superior results for the two larger problems (G532 and Gall) whose sizes necessitated decomposition for TRACS II. Although the ILP of the latest TRACS II version can now solve the largest problem (Rrne) without decomposition, our evolutionary approach has outperformed it in terms of total shift number.

Computational results show that the solutions derived by the new evolutionary approach are very close to that of TRACS II. Compared with all the best known solutions, solution of the SE has 0.24% more shifts in terms of total shift number, and is only 0.74% more expensive in terms of total cost on average. However, our results are much faster in general, especially for larger cases.

6 CONCLUSIONS

Earlier work on the driver scheduling problem based on simplified greedy heuristics sacrifices accuracy for time complexity. Work based on branch-and-bound along with

mathematical programming does the opposite. Recently, researchers have focused on stochastic techniques to get near-optimal solutions within reasonable time. In this paper, we present a novel fuzzy theory based evolutionary approach, which incorporates the idea of fuzzy evaluation into a GA and a SE algorithm, to maintain a balance between accuracy and time complexity. Benchmark experimental results have demonstrated the ability of this evolutionary approach in solving large size real-world driver scheduling problems.

This paper is based on a set covering model for the driver scheduling problem, and as such, it is also relevant to other problems that can be modeled in this way. Furthermore, the idea of using a GA based approach to determine the weights for the fuzzy membership functions may also be applied to the solution of other problems.

Further research is continuing to improve the searching efficiency of the evolutionary approach. In practice, the *Selection* step and the *Mutation* step in SE might be improved by more sophisticated, such as adaptive, operators.

References

Cai, X and Li K.N. (2000) "A Genetic Algorithm for Scheduling Staff of Mixed Skills under Multi-Criteria," European Journal of Operational Research, vol. 125, pp. 141-151.

Chvátal, V. (1979) "A Greedy Heuristic for the Set-Covering Problem," Mathematics of Operations Research, vol. 4, pp. 233-235.

Daduna, J.R., Branco, I. and Paixao, J.M.P. (Eds.) (1995) "Computer-Aided Transit Scheduling," Proceedings, Lisbon, Portugal, Springer-Verlag.

Desrochers, M. and Rousseau J.-M. (Eds.) (1992) "Computer-Aided Transit Scheduling," Proceedings, Montreal, Canada, Springer-Verlag.

Dubois, D. and Prade, H. (1980) "Fuzzy Sets and Systems: Theory and Applications", Academic Press, New York.

Fores, S., Proll, L. and Wren, A. (1999) "An Improved ILP System for Driver Scheduling," in Wilson, N.H.M. (Ed.), Computer-Aided Transit Scheduling, pp. 43-62, Springer-Verlag.

Garey, M.R. and Johnson, D.S. (1979) "Computers and Intractability: a Guide to the Theory of NP-Completeness," Freeman, San Francisco.

Goldberg, D.E. (1989) "Genetic Algorithms in Search, Optimization and Machine Leaning," Addison-Wesley.

Karmarkar, N. (1984) "A New Polynomial-Time Algorithm for Linear Programming," Combinatorica, vol. 4, pp. 373-395.

Kling, R.M. and Banerjee, P. (1987) "ESP: A New Standard Cell Placement Package Using Simulated Evolution," Proceedings of the 24th ACM/IEEE Design Automation Conference, pp. 60-66.

Klir, G. and Yuan, B. (1995) "Fuzzy Sets and Fuzzy Logic," Prentice Hall PTR, New Jersey.

Kwan, A.S.K. (1999) "Train Driver Scheduling," PhD thesis, School of Computing, University of Leeds.

Kwan, A.S.K., Kwan, R.S.K. and Wren, A. (1999) "Driver Scheduling using Genetic Algorithms with Embedded Combinatorial Traits," in Wilson, N.H.M. (Ed.), Computer-Aided Transit Scheduling, Springer-Verlag, pp. 81-102.

Kwan, R.S.K., Wren, A and Kwan, A.S.K. (2000) "Hybrid Genetic Algorithms for Scheduling Bus and Train Drivers," IEEE Congress on Evolutionary Computation, pp. 285-292.

Lin, Y-L., Hsu, Y-C and Tsai, F-S. (1989) "SILK: a Simulated Evolution Router," IEEE Transaction on Computer-Aided Design, vol. 8, pp. 1108-1114.

Lovász, L. (1975) "On the Ratio of Optimal Integral and Fractional Covers," Discrete Mathematics, vol. 13, pp. 383-390.

Ly, T.A. and Mowchenko, J.T. (1993) "Applying Simulated Evolution to High Level Synthesis," IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, vol. 12, pp. 389-409.

Proll, L. (1997) "Stronger Formulations of Mixed Integer Linear Programs: an Example," International Journal of Mathematical Education in Science and Technology, vol. 28, pp. 707-712.

Sait, S.M., Youssef, H. and Ali, H. (1999) "Fuzzy Simulated Evolution Algorithm for Multi-objective Optimization of VLSI Placement," IEEE Congress on Evolutionary Computation, pp. 91-97.

Slavík, P. (1996) "A Tight Analysis of the Greedy Algorithm for Set Cover," ACM Symposium on Theory of Computing, pp. 435-441.

Srinvas, M. and Patnaik, L.M. (1994) "Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms," IEEE Transaction on System, Man and Cybernetics, vol. 24, pp. 656-667.

Wilson, N.H.M. (Ed.) (1999) "Computer-Aided Transit Scheduling," Proceedings, Cambridge, MA, USA, Springer-Verlag.

Wren, A. and Rousseau, J.-M. (1995) "Bus Driver Scheduling – an Overview," in Daduna, J.R., Branco, I. and Paixao J.M.P. (Eds.), Proceedings of the Sixth International Workshop on Computer-Aided Scheduling of Public Transport, pp. 173-187, Springer-Verlag.

Zadeh, L.A. (1965) "Fuzzy Sets," Information and Control, vol. 8, pp. 338-353.

A Reconfigurable Optimizing Scheduler

David J. Montana
BBN Technologies
10 Moulton Street, Cambridge, MA 02138
dmontana@bbn.com

Abstract

We have created a framework that provides a way to represent a wide range of scheduling and assignment problems across many domains. We have also created an optimizing scheduler that can, without modification, solve any problem represented using this framework. The three components of a problem representation are the metadata, the data, and the scheduling semantics. The scheduler performs the optimization using an order-based genetic algorithm to feed different task orderings to a greedy schedule/assignment builder. The scheduler obeys the hard and soft constraints specified in the scheduling semantics. We have applied this reconfigurable scheduler to a variety of scheduling and assignment problems including the job shop, traveling salesman, vehicle routing, and generalized assignment problems. The results demonstrate that the optimizer can provide not only easy reconfigurability but also competitive performance.

1 Introduction

Optimizing schedulers have traditionally targeted a single problem or narrow class of problems. Changing a scheduler to handle a new problem or domain has required redesigning the scheduler and rewriting portions of its software. This introduces an expense that makes optimized scheduling impractical for most applications that could benefit from it. Only applications with large amounts of money tied to the quality of the schedules can justify the costs of developing custom software and algorithms.

Our work aims to change this. We provide a simple yet effective way for a user to configure our optimizing scheduler for a particular problem/domain. Configuring our scheduler does not require recoding or detailed knowledge of how the scheduler works. This can potentially make optimized scheduling sufficiently inexpensive to be practical for a far greater range of problems than it is currently.

Other researchers have recognized the benefits of a unified or reconfigurable approach to scheduling. [Smith and Becker, 1997] creates a unified scheduling ontology, but this ontology is not well suited to simple representation of a problem and is not in a form easily used by an optimizing scheduler. [Davis and Fox, 1994] and [McIlhagga, 1997] both make initial attempts at a reconfigurable scheduler, but they fall short in terms of the generality and flexibility required. The work on AMPL [Fourer *et al.*, 1993] does emphasize easy reconfigurability. It is similar to our approach in its use of algebraic expressions to define the problem as well as its separation of the problem specification from the solver. It is different from our approach because it is targeted at mathematical programming applications and not well suited to many symbolically oriented scheduling problems.

The two key innovations that have allowed us to create a truly reconfigurable optimizing scheduler are in the problem representation. The first is letting the user define the metadata, i.e. the formats for all the data sent to the scheduler. Hence, for any problem the user can define a data representation that is natural for that problem. The second innovation is allowing the user to specify the scheduling semantics using formulas. This allows the scheduler to compute problem-specific information such as whether a resource can perform a task or how much time a resource takes to perform a task.

Our scheduler uses an approach that was introduced by [Whitley *et al.*, 1989] and refined by [Syswerda, 1991]. An order-based genetic algorithm generates task orderings to feed to a greedy schedule builder. What is novel about our scheduler is the way that it can solve any scheduling problem represented using our problem representation framework. Hence, the scheduler is truly reconfigurable.

In the remainder of the paper, we start with an overview of

Constraint	Return Type	Defined Variables	Default Value	Description
Optimization Criterion	number		0	Numerical measure of quality of the current full schedule
Optimization Direction	multiple choice	N/A	minimize	Must be either minimize or maximize
Delta Criterion	number	task, resource	0	Incremental contribution to optimization criterion introduced by having resource perform task
Best Time	datetime	task, resource	starttime	Optimal time for the task to begin
Capability	boolean	task, resource	true	Whether resource has the required skills to perform task
Task Duration	number	task, resource	0	How many seconds it takes resource to perform task
Setup Duration	number	task, previous, resource	0	How many seconds it takes resource to prepare to perform task if it last performed previous
Wrapup Duration	number	task, next, resource	0	How many seconds it takes resource to clean up after doing task if it will be performing next
Prerequisites	list of strings	task	empty list	Names of all the tasks that must be scheduled before scheduling task
Task Unavailability	list of intervals	task, resource, prerequisites	empty list	All intervals of time when task cannot be scheduled (label1 and label2 fields ignored)
Resource Unavailability	list of intervals	resource	empty list	All intervals of time when resource is busy (label1 and label2 can be used for text and color)
Capacity Contribution	list of numbers	task	0	How much task contributes towards filling each type of capacity
Capacity Threshold	list of numbers	resource	0	How much capacity of each type that resource has
Multitasking	multiple choice	N/A	none	Ability of resources to perform more than one task at a time (none, ungrouped, or grouped)
Groupable	boolean	task1, task2	false	Whether task1 and task2 can be placed in the same group

Table 1: List of the various constraints that can be specified

the problem representation framework. We then describe how our scheduler utilizes the information in a problem representation in order to find an optimized schedule for that problem. We conclude with some results on the performance of the scheduler.

2 The Problem Representation Framework

A problem representation consists of three components: the metadata, the data, and the scheduling semantics. We now provide an abbreviated discussion of what each of these involves. More details on the problem representation framework are available in [Montana, 2001].

Metadata - Our scheduling system provides a small number of atomic data types (string, number, boolean, datetime, and list) and predefined composite data types (interval, xycoord, latlong, and matrix). The user builds new composite data types (also called *object types*) from these atomic and predefined types. The data type for a field can itself be another user-defined object, and hence the user can potentially build complex objects. The user must specify a single object type for tasks and a single object type for resources.

Data - Most of the data are instances of object types specified by the metadata. There must be some task instances to schedule and some resource instances to which to assign these tasks. There can also be other data, such as business rules or distance matrices, not associated with a particular task or resource but used as part of the scheduling logic. Two pieces of data that are not object instances are the start and end times of the "scheduling window", which define the earliest and latest time that an assignment can occur. Other non-object data is that specifying which set of assignments from a previously produced schedule should remain frozen in the current scheduler run. (This concept of freezing is important for dynamic rescheduling.)

Scheduling Semantics - We have defined a set of general constraints that define what constitutes a legal and optimized schedule. For most of these constraints, the user specifies a formula that tells how to compute the value of the constraint in a given context. For example, the Task Duration constraint tells how many seconds it takes a particular resource to perform a particular task. If this value is

obtained by dividing the distance field of the task object by the speed field of the resource object, then the formula to specify for this constraint is task.distance / resource.speed. A description of the mini-language for specifying formulas in given in [Montana, 2001]; the examples in Section 3 should provide an idea of how these formulas work and what they can express.

Table 1 lists all the different constraints for which the user can specify a formula. If the user does not specify a formula, the default value is used. The context in which the constraint is evaluated is given by the value of the variables that are defined. While some variables (tasks, resources, starttime, and endtime) are defined for all constraints, some variables (e.g., task and resource) are defined only for certain constraints. The descriptions provided are brief; Section 4 provides a better understanding of some of these constraints by describing how they are actually used.

3 Examples of Problem Specifications

We now describe four examples of problem specifications. These well-known problems from the operations research literature are the problems we used for the experiments described in Section 5. (The OR-Library [Beasley, 1990] is a good source of such classic problems.) We have specified and solved problems much more algorithmically complex than those given here, but these highly idealized problems provide a good introduction to how to specify a problem.

3.1 Traveling Salesman Problem (TSP)

There is a salesman who needs to start at a given city, travel to a set of other cities visiting each city once, and then return to the starting city. The distance from any city to any other city is provided. The objective is to minimize the total distance traveled.

The task object, *city*, and resource object, *salesman*, are defined to have the fields:

- **city** - id (string) and index (number)
- **salesman** - id (string)

There is one salesman with arbitrary id; N cities with index = i and id = "City i" for $i = 1, ..., N$; and an NxN matrix named *distances* that contains all the intercity distances. For the scheduling semantics, the constraints with non-default values are shown in Table 2.

Constraint	Formula
Optimization Criterion	maxover (resources, "r", complete (r)) - starttime
Setup Duration	matentry (distances, task.index, if (hasvalue (previous), previous.index, 1))
Prerequisites	if (task.id = "City 1", mapover (tasks, "t2", if (t2.id != "City 1", t2.id)))

Table 2: Constraints for Traveling Salesman Problem

Constraint	Formula
Optimization Criterion	sumover (resources, "r", preptime (r)) + sumover (tasks, "t", if (hasvalue (resourcefor (t)), 0, 1000))
Delta Criterion	preptime (resource) - previousdelta (resource)
Task Duration	extra.servicetime
Setup Duration	dist (task.location, if (hasvalue (previous), previous.location, extra.depotlocation))
Wrapup Duration	if (hasvalue (next), 0, dist (task.location, extra.depotlocation))
Task Unavailability	list (interval (starttime, starttime + task.earliest), interval (starttime + task.latest + extra.servicetime, endtime))
Capacity Contributions	list (task.load)
Capacity Thresholds	list (resource.capacity)

Table 3: Constraints for Vehicle Routing Problem

3.2 Vehicle Routing Problem with Time Windows

This problem is described in [Solomon, 1987]. There are M vehicles and N customers from whom to pick up cargo. Each vehicle has a limited capacity for cargo, and each piece of cargo contributes a different amount towards this capacity. There is a certain window of time in which each pickup must be initiated, and the pickups require a certain non-zero time. Each vehicle that is utilized starts at a central depot, makes a circuit of all its customers, and then returns to the depot. The objective is to minimize the total distance traveled by the vehicles.

The problem-specific objects are:

- **customer** - id (string), load (number), earliest (number), latest (number), and location (xycoord)
- **vehicle** - id (string) and capacity (number)
- **extradata** - servicetime (number) and depotlocation (xycoord)

The single object of type extradata is named *extra*. For the scheduling semantics, the constraints with non-default values are shown in Table 3.

3.3 Generalized Assignment Problem (GAP)

This problem is describe in [Osman, 1995]. There are N jobs to be assigned to M agents. There are defined assignment costs, one associated with each pairing of a job and

Constraint	Formula
Optimization Criterion	sumover (tasks, “t”, entry (t.costs, resourcefor (t).index))
Optimization Direction	maximize
Delta Criterion	entry (task.costs, resource.index)
Capacity Contributions	task.loads
Capacity Thresholds	loop (length (resources), “i”, if (i = resource.index, resource.capacity, 100000))

Table 4: Constraints for Generalized Assignment Problem

an agent. Each agent has a defined capacity, and each job contributes a defined amount towards the capacity of each agent, with this amount depending on the agent. The objective is to maximize the total costs.

The problem-specific objects are:
- **job** - id (string), index (number), costs (list of numbers), and loads (list of numbers)
- **agent** - id (string), index (number), and capacity (number)

The costs field of each job contains one cost for each agent, which can be accessed from the list using the index of the agent. The same applies to the loads field of each job. For the scheduling semantics, the constraints with non-default values are shown in Table 4.

3.4 Job-Shop Scheduling Problem (JSSP)

This problem was originally proposed by [Muth and Thompson, 1963]. There are M machines and N manufacturing jobs to be completed. Each job has M steps, with each step corresponding to a different specified machine. There is a specified order in which the steps for a certain job must be performed, with one step not able to start until the previous step has ended. The objective is to minimize the end time of the last step completed.

The problem-specific objects are:
- **step** - id (string), duration (number), machine (string), and preceedingstep (string)
- **machine** - id (string)

For the scheduling semantics, the constraints with non-default values are shown in Table 5.

4 The Reconfigurable Scheduler

We have created a scheduler that is capable of finding an optimized solution for any scheduling problem specified using the framework described above. A “greedy”, i.e. locally optimal, scheduler builder takes a particular ordering of tasks and assigns them one at a time to the best resource for that task. A genetic algorithm generates different task orderings to feed the greedy schedule builder, searching for an optimal ordering. The overall control flow of the scheduler is shown in Figure 1.

Constraint	Formula
Optimization Criterion	maxover (resources, “r”, complete (r)) - starttime
Capability	task.machine = resource.id
Task Duration	task.duration
Prerequisites	if (task.preceedingstep != “”, list (preceedingstep))
Task Unavailability	mapover (prerequisites, “t”, interval (starttime, taskendtime (t)))

Table 5: Constraints for Job-shop Scheduling Problem

```
Greedy initialization;
Genetic loop:
  Determine new task ordering;
  Task (greedy) loop:
    Find next task to schedule;
    Resource (greedy) loop:
      Find next capable resource;
      Time (greedy) loop:
        Search to find best interval
         for resource to perform task;
      end Time loop;
      Check whether this
       resource/interval best so far;
    end Resource loop;
    Assign task to best resource
     during best interval;
  end Task loop;
  Evaluate fitness of schedule;
end Genetic loop;
```

Figure 1: Control flow of the scheduler

4.1 The Genetic Algorithm

The genetic algorithm is a fairly standard order-based one. We number each task from 1 to N, where N is the number of tasks, and a chromosome is some permutation of the numbers 1 through N. The crossover operator we use is position-based crossover, which is described in [Syswerda, 1991]. The mutation operator is a variation on Syswerda’s order-based mutation except that, instead of selecting only two positions whose order to exchange, our mutation selects between 2 and N positions whose order is randomly generated while the other positions remain the same. The

population is initialized by choosing random orderings.

The replacement scheme is steady-state rather than generational, i.e. a single child enters the population and the worst individual leaves the population in a single "generational cycle". Duplicate individuals are not allowed in the population. The parent selection probabilities are exponentially distributed. The parameter parent-scalar is defined as the ratio of the probabilities of selecting the i^{th} best individual and of selecting the $(i-1)^{st}$ best individual.

There are four conditions under which the genetic algorithm can terminate. First, it will stop if the elapsed wall time of its current run exceeds a parameter (max-time). Second, it will terminate if the total number of evaluations (i.e., individuals generated) exceeds a parameter (max-evals). Third, it will stop if the best score has not improved for a consecutive number of evaluations exceeding a parameter (max-top-dog-age). Fourth, it will terminate if the number of duplicate individuals generated exceeds a parameter (max-duplicates).

Evaluation of an individual is done by first feeding the ordering of the tasks to the greedy schedule builder and letting it build a schedule. The formula given by the Optimization Criterion constraint is then executed on this schedule. The number returned by the formula is the chromosome's fitness.

4.2 The Greedy Schedule Builder

The algorithm of the greedy schedule builder, although simple in concept, is complicated by the need to consider so many different factors. For the special case of the job-shop scheduling problem, our greedy scheduler is equivalent to the active schedule generation algorithm presented in [Giffler and Thompson, 1960]. However, to handle problems other than the job-shop problem, our greedy scheduler must consider a variety of other factors including:

- resource selection - Many scheduling problems allow a choice between different qualified resources for each task.
- time selection - For many scheduling problems finishing a task earlier is not always better, such as is the case with just-in-time scheduling.
- multitasking - Some scheduling problems allow resources to perform more than one task simultaneously.

As shown in Figure 1 there are different components of the greedy schedule builder. We now discuss each of these.

Initialization - There are certain results that the greedy schedule builder needs but that do not vary based on what assignments are made. For the sake of efficiency, these are computed once before the genetic algorithm even starts. These results include:

- **Lists of capable resources -** For each task, it creates a list of all those resources that have the skills/capabilities to perform that task. It determines whether a resource has the required skills by executing the Capability formula with the *task* variable set to the task and the *resource* variable set to the resource.
- **Resource unavailable times -** For each resource, it computes a set of nonoverlapping intervals of time for which that resource is not available to be assigned to a task due to other commitments (e.g., time off or maintenance). To do this, it executes the Resource Unavailable Times formula with the *resource* variable set appropriately to obtain a preliminary set of intervals. It adds to this list the intervals that represent the constraint that resources should not be scheduled before the start or after the end of the scheduling window of the window. Then, it resolves these into a set of nonoverlapping intervals.
- **Capacity contributions -** For each task, it computes the task's contribution towards each of the capacities by executing the Capacity Contributions formula with the *task* variable set appropriately. The i^{th} element of the list is the contribution to the i^{th} capacity.
- **Capacity thresholds -** For each resource, it computes the resource's threshold for each of the capacities using the Capacity Thresholds formula.
- **Prerequisites -** For each task, it computes the set of other tasks that must be scheduled prior to this task regardless of the ordering of tasks provided by the genetic algorithm. The Prerequisites formula provides a list of task names, which are used to look up the task objects.

Task Loop - The greedy schedule builder assigns one task at a time. It attempts to adhere as much as possible to the order in the chromosome, but it will not schedule a task before its prerequisites have been scheduled. So, each time through the loop it picks the task earliest in the chromosome that has not yet been scheduled but all of whose prerequisite tasks have been scheduled. After executing the resource loop in order to find the best resource and time, it assigns the task to that resource at that time. If there is no resource that is capable and available to perform the task, then the task is marked as unassigned.

The assignment process involves the following steps. First, the task must be inserted into the resource's schedule. If the Multitasking selection is grouped and the resource loop has specified a particular group for the task, then the task is placed in this group. Otherwise, a new schedule entry is made for this task and resource with setup start time, task start time, task end time, and wrapup end time as specified from the resource loop. (The time interval associated with a task assignment is divided into three consecutive intervals: the setup interval when the resource is preparing to perform the task, the task interval when the resource performs the task, and the wrapup interval when the resource

cleans up. The four times represent the boundaries of these three intervals.) The wrapup end time of the previous task in the resource's schedule and the setup start time of the next task are also updated if necessary as specified by the resource loop. If there is grouped multitasking, then a new entry is also a new group.

Next, the capacities are updated. If the Multitasking selection is none, the capacities are single aggregates summed over time, and the capacities used by the resource are updated by adding the capacity contributions from the task. Otherwise, the capacities are time histories, and they are updated accordingly.

Resource Loop - To find the best resource and interval of time to which to assign a given task, the greedy schedule builder examines each resource on the task's list of capable resources. For a given resource, it starts by computing, using the Best Time formula, the ideal time for the task start time. This is a soft constraint that tells the time loop where to start its search. It also computes two hard constraints on time, the task duration and the task unavailable times, using the corresponding formulas. It then uses the time loop to search forward from the best time for the nearest legal task start time, where a time is legal if

- the resource is available for the entire interval between the corresponding setup start time and wrapup end time, and the task is available between the task start time and the corresponding task end time
- the setup start time for the task is not earlier than the wrapup end time from the previous task for that resource, and the wrapup end time of the task is not later than the setup start time of the next task
- none of the aggregate capacity contributions exceed their corresponding capacity thresholds

Alternatively, if there is grouped multitasking, then a task start time is legal if it is the task start time for an existing group such that

- its task duration is no longer than the task duration of the group
- the aggregate capacity contributions of the group after adding the task do not exceed any capacity thresholds
- executing the Groupable formula for this task and a task already in the group returns true

If the forward search yields a legal time, then it makes a temporary assignment of the task to the resource at the specified time, and evaluates the Delta Criterion formula to obtain a fast measure of how good that assignment would be. If the forward search yields no time or a time which is not the best time, then it repeats the process, this time searching backward from the best time for the closest legal start time. If neither the forward or backward search yields a time, then the task cannot be assigned to this resource. If the forward and backward search both yield times, then it picks the one with the best delta criterion. The resource (and time) with the best delta criterion is selected for assignment.

Time Loop - When performing the search for the legal task start time closest to the best time, there are a few items about which to be careful. First, the setup and wraup durations depend respectively on the previous and next task in the resource's schedule. Hence, they can only be computed in the context of a proposed position of the task in the resource's schedule. Additionally, the previous task's wrapup time and next task's setup time (if these tasks exist) are potentially altered by the placement of the new task and must therefore be recomputed. All these quantities are stored along with the task start time to allow the task loop to make the assignment. A second item to be careful about is that this is the innermost loop and hence is executed the most frequently. Therefore, it needs to be particularly efficient.

5 Experimental Results

The data for which we have executed our experiments are instances of the problems given in Section 3. These are commonly studied problems that we use because they allow comparison with other algorithms. We cannot hope to match the performance of the best algorithms developed for these problems for two reasons. First, we do not tune our algorithm to any particular problem and therefore will generally not achieve optimal performance for a particular problem. Second, the formulas are not compiled directly into machine code but rather are interpreted, and hence they execute less efficiently than compiled code. However, the benefit of our approach is the wide range of problems it can handle and the ease with which it can handle new problems, so we only need to prove reasonably good, not optimal, performance.

For each experiment, we have selected a particular data set and a particular set of genetic algorithm parameters, and we have made ten genetic scheduler runs. Table 6 summarizes the results of these experiments. Note that for each experiment, Table 6 tells the key genetic algorithm parameters: population size, parent-scalar, and either max-evals or max-top-dog-age (depending on which actually caused all the terminations). The table also gives the following results from the experiments:

- Best Known Score - the score of either the provably best solution or the best solution found by any algorithm to date (used as a reference)
- Best Score - the score of the best solution from all ten runs
- Median Score - the median of the scores of the ten solutions found by the ten runs
- Average Score - the mean of the scores from the ten runs

- Average Number of Evaluations - the average number of individuals evaluated in a run before the run terminated (because the genetic algorithm is steady-state, this is a better measure than the number of generations)
- Average Time Per Run - the average amount of time it required a run to execute to completion
- Time Per Evaluation - the average number of milliseconds required to perform a single evaluation

All the runs were made on a 200 MHz UltraSparc processor.

For the traveling salesman problem, we have so far used a single instance, bays29, which is a 29-city symmetric problem available at the TSPLIB web site. The first two rows in Table 6 correspond to two sets of runs for this data with different genetic algorithm parameters. The first row has a larger population, proportionately lower fitness pressure from parent-scalar and a larger max-top-dog-age. It does well at finding nearly optimal solution. The second row runs faster but does not do as well. This illustrates the tradeoff between search time and quality of solution. (A third factor in the tradeoff is computational power and its cost, particularly with an inherently parallelizable algorithm such as a genetic algorithm.) This is a relatively small traveling salesman problem, and while we could practically do significantly bigger problems, this algorithm cannot compete with specially designed algorithms such as [Lin and Kernighan, 1973].

For the job-shop scheduling problem, we have so far used only the Muth-Thompson 6x6 data [Muth and Thompson, 1963], referred to as ft06 at the OR-Library web site. It contains 36 tasks and 6 resources. Despite the fact that this is larger than the traveling salesman problem, the scheduler clearly has an easier time with the job-shop problem. The time per evaluation is roughly the same even though the job-shop problem has more resources because the job-shop problem has only one capable resource per task, and that is a better measure of the computation required. The jobshop problem requires less evaluations to find the optimal solution because the search space is in practice smaller. This is because the constraints in the job-shop problem, particularly the prerequisites constraint, make it so that many different chromosomes decode to the same schedule. One lesson is that one cannot predict the search time required purely based on the number of tasks and resources.

The generalized assignment problem is so far the only problem for which we have experimented with multiple instances. From the OR-Library web site, we have used c515-1 (5 resources and 15 tasks), c530-1 (5 resources and 30 tasks), and c1030-1 (10 resources and 30 tasks). This has allowed a very preliminary examination of the scaling properties of our algorithm. We would expect the time per evaluation to be roughly proportional to the product of the number of tasks and the number of capable resources per task (which in this case is the number of resources), and this is the case for this data. We would also expect an increase in the number of evaluations required with an increase in the number of tasks due to the larger search space, and this is also borne out by the data. Overall, these problems are solved quickly because the greedy algorithm does most of the work. One interesting result is that while the algorithm can get close to the optimal solution for c530-1 quickly, it requires a long search to find the best solution.

The next logical step for the experimentation process is to perform the same experiments for larger search problem such as the Muth-Thompson 10x10 job-shop problem or the Solomon vehicle routing problems.

6 Conclusions and Future Work

We have developed a powerful framework for representing scheduling problems, and we have built a reconfigurable scheduler that can find an optimized solution for any problem specified in this framework. The optimization performance of this scheduler is good, even though the generality of our approach does mean that, for certain problems, we cannot acheive the performance a scheduler designed specifically for that problem. The major benefit of reconfigurability is that it makes development of optimized scheduling for a wide range of problems simple and inexpensive. There is a vast array of scheduling problems that are currently solved using manual or non-optimized scheduling, and for most of these problems making optimized scheduling practical requires a simple and inexpensive solution rather than the best possible performance.

Further enhancing the ease of use of our reconfigurable scheduler is a web-based system we have built to allow the user to interact with the scheduler. The details of this interface are beyond the scope of this paper, but in general terms the browser-based interface allows the user to fully specify a problem (metadata, data, and scheduling semantics), start a new scheduler run and check on its progress, and graphically view the schedules. Using display constraints similar to the scheduling constraints described in Section 2 allows the user to select the colors and text to display with each assignment.

Also beyond the scope of this paper but illustrating the advantages of reconfigurability, we have integrated our reconfigurable scheduler into the same multiagent infrastructure as described in [Montana *et al.*, 2000]. This has allowed us to build multiagent societies that have included multiple interacting reconfigurable scheduling agents as well as other types of agents.

There are two directions in which to extend our work on the reconfigurable scheduler. First, as we expand the prob-

Problem Name	Pop Size	Parent Scalar	Max Evals	Max Top Dog	Best Known Score	Best Score	Median Score	Avg Score	Avg Num Evals	Avg Time (M:S)	Msecs Per Eval
TSP-bays29	5000	0.998	N/A	20000	2020	2028	2028	2042	134,429	13:25	5.99
TSP-bays29	1000	0.99	N/A	4000	2020	2058	2204	2191	26,680	2:41	6.02
JSSP-mt06	1000	0.99	5000	N/A	55	55	55	55	5000	0:54	10.9
GAP-c515-1	500	0.98	2500	N/A	336	336	336	336	2500	0:09	3.48
GAP-c1030-1	1000	0.99	8000	N/A	709	709	709	708.8	8000	1:31	11.4
GAP-c530-1	1000	0.99	5000	N/A	656	655	653	653.3	5000	0:39	7.88
GAP-c530-1	20000	0.9995	100000	N/A	656	656	656	655.3	100000	14:10	8.50

Table 6: Summary of experimental results

lem representation, we need to extend the scheduler capabilities to match. Currently, the problem representation framework does not allow certain concepts such as resettable capacities (e.g., the ability to empty a load) or multiple resources per task. When we put these into the problem representation, the scheduler algorithm needs to handle them. Second, we should make the scheduler smarter about handling special cases. If the scheduler could recognize special cases, then it could apply special-purpose, higher-performance algorithms for these cases. This would improve the performance of the scheduler without sacrificing its generality.

Acknowledgments

This work was supported by DARPA contract MDA972-97-C-0800 under the Advanced Logistics Program. Thanks to Gordon Vidaver for his ideas and his help coding and to Todd Carrico for challenging us to build a generic scheduler.

References

[Beasley, 1990] J. Beasley. OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

[Davis and Fox, 1994] G. Davis and M. Fox. ODO: A constraint-based architecture for representing and reasoning about scheduling problems. In *Proceedings of the 3rd Industrial Engineering Research Conference*, 1994.

[Fourer *et al.*, 1993] R. Fourer, D. Gay, and B. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, 1993.

[Giffler and Thompson, 1960] B. Giffler and G. Thompson. Algorithms for solving production-scheduling problems. *Operations Research*, 8(4):487–503, 1960.

[Lin and Kernighan, 1973] S. Lin and B. Kernighan. An effective heuristic algorithm for the TSP. *Operations Research*, 21(2):498–516, 1973.

[McIlhagga, 1997] M. McIlhagga. Solving generic scheduling problems with a distributed genetic algorithm. In *Proceedings of the AISB Workshop on Evolutionary Computing*, pages 85–90, 1997.

[Montana *et al.*, 2000] D. Montana, J. Herrero, G. Vidaver, and G. Bidwell. A multiagent society for military transporation scheduling. *Journal of Scheduling*, 3(4):225–246, 2000.

[Montana, 2001] D. Montana. A problem representation framework for a reconfigurable scheduler, 2001. Currently unpublished.

[Muth and Thompson, 1963] J. Muth and G. Thompson. *Industrial Scheduling*. Prentice Hall, 1963.

[Osman, 1995] I. Osman. Heuristics for the generalised assignment problem: Simulated annealing and tabu search approaches. *OR Spektrum*, 17:211–225, 1995.

[Smith and Becker, 1997] S. Smith and M. Becker. An ontology for constructing scheduling systems. In *Working Notes of 1997 AAAI Symposium on Ontological Engineering*, 1997.

[Solomon, 1987] M. Solomon. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, 35:254–165, 1987.

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

[Whitley *et al.*, 1989] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 133–140, 1989.

A Hybrid Genetic Algorithm for the Generalized Traveling Salesman Problem

Jean-Yves Potvin and **Daniel De Ladurantaye**
Centre de recherche sur les transports
Université de Montréal,
C.P. 6128, succursale Centre-ville,
Montréal, Québec, Canada H3C 3J7

Abstract

The Generalized Traveling Salesman Problem consists of determining a shortest tour on a graph passing through each of several clusters of vertices. A hybrid genetic algorithm (GA) is developed to solve a variant of this problem where exactly one vertex must be visited in each cluster. In this algorithm, the GA searches for a good selection of vertices, while classical operations research techniques are used to produce a tour with the selected vertices. Numerical results are reported on a standard set of benchmark problems and a comparison is provided with the two best heuristics reported in the literature.

1 Introduction

The symmetric Traveling Salesman Problem (TSP) is a canonical NP-hard problem in combinatorial optimization [8]. Given a complete undirected graph with a length associated with each edge, the objective is to find a shortest tour passing through each vertex exactly once. Such a tour is also known as a Hamiltonian cycle. The Generalized TSP (GTSP) extends the classical problem by partitioning the set of vertices into a number of subsets or clusters. Then, a shortest tour passing through at least one vertex in each cluster must be found. Note that when the number of subsets is equal to the number of vertices (i.e., when there is only one vertex in each cluster), the GTSP reduces to the TSP. Applications of the GTSP are reported in many areas such as location-routing, computer design, loop material flow system design and postal box collection [7].

Different variants of the GTSP are found in the literature. Here, we consider problems defined on the Euclidean plane, and where exactly one vertex must be visited in each cluster. More formally let $G = (V, E)$ be an undirected graph where $V = \{1, \ldots, n\}$ is the vertex set, $E = \{(i, j) : i, j \in V, i < j\}$ is the edge set, and a non negative length or distance d_{ij} is associated with every edge (i, j). If the set V is partitioned into m clusters $V_1, \ldots, V_m$, the problem is to find a shortest cycle which contains exactly one vertex in each cluster. Exact algorithms to solve this problem, using branch-and-cut or dynamic programming, are found in [4, 6, 15, 17]. Due to their exponential nature, however, these approaches are restricted to relatively small-sized instances. Recent heuristic approaches are reported in [1, 13, 16].

In the following, Section 2 first presents issues related to solution representation and fitness evaluation for a genetic algorithm (GA). Section 3 then describes the hybrid GA proposed for solving the GTSP problem. Finally, Section 4 reports computational results obtained on a set of benchmark problems. The conclusion follows.

2 Solution representation

Genetic algorithms have been widely used for solving different combinatorial optimization problems, including the TSP [11]. But, to the best of our knowledge, this is the first application of a GA to the GTSP. The procedure that we propose hybridizes the GA with classical operations research techniques. Basically, the GA searches for a good selection of vertices, one in each cluster, while classical operations research techniques are used to produce a solution with the selected vertices (i.e., a good ordering of these vertices on the tour).

When applying a GA to a combinatorial optimization problem, an appropriate represention must first be chosen, given that classical bit-strings are often inap-

propriate. In the following, the chosen representation is introduced and the way to decode it into a solution of the GTSP is explained.

2.1 Encoding

Here, a chromosome is a string of m integers, where the integer in position i corresponds to the vertex selected in cluster V_i, $i = 1, ..., m$. Thus, each chromosome simply represents a set of m vertices, and their ordering is irrelevant at this stage. The true ordering is done subsequently, using operations research techniques (see Section 2.2).

2.2 Decoding

The chromosome is decoded into a solution of the GTSP by ordering its vertices and its fitness corresponds to the value of the solution produced (i.e., the tour length). To order the set of vertices, the farthest insertion heuristic is first applied. Then, local search heuristics based on edge exchanges are performed for further improvement. These methods are briefly described in the following.

Farthest insertion heuristic. The Farthest Insertion (FI) heuristic inserts the vertices one by one in the tour. The next vertex to be inserted is the one which maximizes the minimum distance to the vertices already included in the tour. Then, the insertion place is chosen to minimize the detour. More precisely:

1. Select a vertex i at random among the m vertices.
2. Select vertex k which is the farthest from vertex i and form the subtour $i - k - i$.
3. Select vertex k not in the subtour which is the farthest from the vertices in the subtour.
4. Find edge (i, j) in the subtour such that the detour $d_{ik} + d_{kj} - d_{ij}$ is minimal. Insert k between i and j.
5. If all vertices have been inserted, STOP. Otherwise, go to step 3.

It is worth noting that the complexity of FI is $O(m^2)$.

Edge exchanges. The 2-opt [9], Or-opt [10] and 4-opt* [14] local search heuristics are applied one by one, in this order, to the current solution for further improvement. In each case, the search framework is the following:

1. Start with the tour s produced by FI (in the case of 2-opt) or the previous local search heuristic (in the case of Or-opt and 4-opt*).
2. Generate the neighbors of s and select the best one s'.
3. If s' is worse than s, then STOP with s. Otherwise $s \leftarrow s'$ and go back step 2.

The neighborhood of s in step 2 depends on the edge exchange heuristic. In the case of *2-opt*, new solutions are produced by replacing two edges in the current solution by two new edges. This neighborhood is of complexity $O(m^2)$ as it corresponds to the number of ways to select two edges to be removed among m edges. The *Or-opt* considers a subset of 3-opt exchanges, where three edges are replaced by three new ones. Basically, a string of one, two or three consecutive vertices is removed from the current tour and reinserted at another place in the tour. Although the 3-opt neighborhood is of complexity $O(m^3)$, Or-opt is of complexity $O(m^2)$ because it considers only a restricted subset of 3-opt exchanges. Finally, the *4-opt** neighborhood corresponds to a subset of 4-opt exchanges, where four edges are replaced by four new ones. Although the complexity of the 4-opt neighborhood is $O(m^4)$, 4-opt* is of complexity $O(m^2)$ because stringent conditions must be satisfied for an exchange to be valid. In particular, an exchange should not lead to the displacement of a string of customers with a length exceeding a given threshold. This heuristic is rather complicated and the interested reader is referred to [14] for details.

In the next section, the algorithmic framework of the GA is presented and each of its components is described in turn.

3 The algorithm

The algorithmic framework is quite straightforward and can be summarized as follows:

1. Generate an initial population of p solutions.
2. For I generations do:
   2.1 Selection;
   2.2 Crossover;
   2.3 Mutation.
3. Output the best solution found.

3.1 Initial population

The initial population is randomly generated. That is, each new chromosome is created by randomly selecting one vertex in each cluster.

3.2 Selection

Before selecting the parents, a rank-based method is first used to associate a value with each chromosome [3, 19]. The chromosome with highest fitness gets rank 1 and is assigned some predefined MAX value; the chromosome with lowest fitness gets rank p and is assigned some predefined MIN value. In general, the chromosome of rank i is assigned a value v_i between MIN and MAX based on the following formula

$$v_i = MAX - \frac{(MAX - MIN)(i-1)}{p-1} \qquad (1)$$

In our experiments MAX is set to 1.5 and MIN to 0.5, so that the summation over all chromosome values corresponds to the size of the population. Consequently, the selection probability of a chromosome of rank i is

$$p_i = \frac{v_i}{\sum_{i=1}^{p} v_i} = \frac{v_i}{p} \qquad (2)$$

and the expectancy E_i over p selection trials is simply equal to the value v_i, namely:

$$E_i = p\frac{v_i}{p} = v_i. \qquad (3)$$

Note that this approach does not put any emphasis on the magnitude of the fitness gap between two chromosomes; only their relative order is important. In particular, no special emphasis is put on a dominant chromosome with a very high fitness, thus alleviating premature convergence of the population. Once the selection probabilities have been determined through this rank-based method, the parents are selected using Stochastic Universal Sampling (SUS) [2]. As opposed to the classical roulette-wheel selection, this approach provides a lower and an upper bound of $\lfloor v_i \rfloor$ and $\lceil v_i \rceil$, respectively, on the number of selections for the chromosome of rank i.

Once the parents have been selected, they are then processed by the crossover and mutation operators to produce offspring which encode a new choice of vertices.

3.3 Crossover

Here, two parent chromosomes are chosen at random and mated to produce an offspring. This is repeated until p offspring are produced. Given that chromosomes represent sets of vertices, where the ith position encodes the vertex selected in cluster V_i, classical crossover operators can be easily applied. In our experiments, uniform crossover was used [18] where, at each position (cluster), a parent is randomly selected to provide its vertex to the offspring.

3.4 Mutation

In classical GAs, mutation is often considered as a secondary operator aimed at slightly perturbing the search. In our application, however, mutation was found to be a fundamental operator. The simple random mutation schemes that we first developed, in the spirit of classical GAs, never led to implementations that were even close to the best GTSP heuristics. Only the more sophisticated mechanism presented below allowed us to produce competitive results.

The basic mutation mechanism, called mutation *M1*, processes the chromosome position by position (cluster by cluster) and randomly replaces the selected vertex by another one in the same cluster. This procedure has been integrated within a local search scheme to produce mutation *M2* as follows:

1. Set chromosome c as the initial chromosome.
2. While there is an improvement do:

   For $i = 1, .., m$ do:

   2.1 Replace the vertex at position i in chromosome c by a randomly chosen vertex in cluster V_i to produce chromosome c'.

   2.2 Evaluate the impact of this replacement on solution quality.

   2.3 If the solution associated with chromosome c' is better than the one associated with chromosome c, then $c \leftarrow c'$.
3. Output chromosome c.

In Step 2.2, the impact of the move on solution quality is evaluated as follows. In the tour associated with chromosome c, the vertex in cluster V_i is directly replaced by the randomly chosen one to produce a new tour. We then reoptimize it with the 2-opt, Or-opt and 4-opt* edge exchange heuristics. Note that the tour associated with chromosome c is already locally optimal with regard to these exchange heuristics. Consequently, only a few iterations are needed to reach a new local optimum after the replacement of a single vertex. This is much less expensive than recomputing a new solution "from scratch", by applying the FI heuristic to construct an initial tour and then by reoptimizing this initial tour with 2-opt, Or-opt and 4-opt*.

4 Computational Results

To test our algorithm, we used the 36 benchmark problems of Fischetti et al. [4]. These problems are derived from the TSPs found in the TSPLIB library [12] by applying a clustering procedure which partitions the set of vertices into $m = \lceil \frac{n}{5} \rceil$ clusters. The code was written in C++ and the tests were run on a PC equipped with a Pentium II processor (300 MHz).

The results of our GA are compared with those obtained with the GI^3 heuristic [13] and the tabu search heuristic in [16], which are the best heuristics known to date to solve the GTSP. GI^3 is an insertion heuristic, followed by a local reoptimization procedure based on 2-opt and 3-opt edge exchanges. In this algorithm, the insertion and edge exchange moves are "generalized" to consider different choices of vertices. The algorithm in [16] is more powerful as it uses the mechanisms at the core of the tabu search heuristic [5] to escape from local optima. The neighborhood structure is based on the addition and removal of vertices, and allows the exploration of the infeasible domain through penalties in the objective (i.e., solutions with no vertex or more than one vertex in a given cluster are considered).

4.1 Parameter sensitivity

We performed a number of preliminary experiments on randomly generated problems with up to 500 vertices to evaluate the sensitivity of the solutions produced to various parameter values. Our observations are summarized below.

Population size and number of generations. The population size and number of generations were set at 50 and 400, respectively. On the largest problems, we observed a fast improvement in the first 50 to 100 generations, and then a slower improvement up to generation 300-400, approximately. Increasing the number of generations further did not lead to any significant improvements.

Mutation rate. A global mutation rate p_M must first be defined. Then, mutations *M1* and *M2* are applied using probabilities p_{M1} and p_{M2} with $p_{M1} + p_{M2} = 1$. With regard to the global mutation rate, the best results were obtained with $p_M = 0.5$. This is much higher than in classical GA implementations, where a very small mutation rate is often suggested. We also found that varying the probabilities p_{M1} and p_{M2} during the search, rather than keeping constant values, was beneficial. Basically, a higher probability should be associated with *M1* at the start of the search, and a lower probability towards the end. Mutation *M2* is very useful to get competitive results, but its probability should be kept low at the start to avoid premature convergence. In the current implementation, p_{M1}= 0.8 and p_{M2}= 0.2 for 90% of the iterations; these values are then switched to p_{M1}=0.2 and p_{M2}=0.8 for the last 10% of the iterations. More gradual adjustments to these values may be beneficial, but we did not find a formula that produced significantly better results.

Crossover rate. The crossover rate was set to the standard value of 0.6. Smaller values degraded the solutions, while larger values did not significantly impact solution quality.

4.2 Comparison on benchmark problems

Table 1 compare the results produced by our GA with the results reported in [13] and [16] for GI^3 and TABU, respectively. The number at the end of a problem identifier indicates the size of the problem (e.g., EIL51 contains 51 vertices). The column *Best* refers to the best of the three runs, while *Avg.* is the average. In the case of GI^3, the results reported by Renaud and Boctor in [13] correspond to a single run on each problem instance because there is no stochastic element in their implementation. The values shown are the ratio of the heuristic solution on the optimal one. Therefore, a value of 1.0000 indicates that an optimal solution was found. The column *CPU* is the computation time in seconds. Note that TABU was run on a SUN Sparc 5 and GI^3 on a SUN Sparc LX. Consequently, their CPU times should be divided by 2 and 4, respectively, for a fair comparison with our 300 Mhz PC.

Table 1 shows that our GA implementation is competitive, as it produces solutions within 1% of the optimum on average, like the two other methods. The heuristic GI^3 is the fastest, but leads to solutions that are at (almost) 1% above the optimum. TABU can still be considered as the best approach, since it is faster than GA and generates better solutions, on average. GA exhibits a slightly larger variance from one run to another, as compared with TABU, but this characteristic seems to be beneficial. When the best solution over 3 runs is taken, our algorithm finds a larger number of optimal solutions on the test set (i.e., 24 optimal solutions versus 21) while the average solution values of GA and TABU become very close (i.e, .26% above the optimum for GA versus .20% for TABU, a gap of .06% only). Note that these results were not significantly improved by increasing the number of generations in the GA. For example, after 700 generations, the average percent over the optimum for the best of 3 runs was stable at .26%, with one additional optimal solution found on problem EIL101.

Problem	GA			$TABU$			GI^3	
	Best Length	Avg. Length	CPU	Best Length	Avg. Length	CPU	Length	CPU
EIL51	1.0000	1.0000	3	1.0000	1.0000	17	1.0000	1
ST70	1.0000	1.0010	7	1.0000	1.0000	26	1.0000	2
EIL76	1.0000	1.0000	10	1.0000	1.0000	28	1.0000	2
PR76	1.0000	1.0000	10	1.0000	1.0000	27	1.0000	3
RAT99	1.0000	1.0000	19	1.0000	1.0000	65	1.0000	5
KROA100	1.0000	1.0000	21	1.0000	1.0000	42	1.0000	7
KROB100	1.0000	1.0000	20	1.0000	1.0000	45	1.0000	6
KROC100	1.0000	1.0000	19	1.0000	1.0000	39	1.0000	7
KROD100	1.0000	1.0000	21	1.0000	1.0000	39	1.0000	9
KROE100	1.0000	1.0032	20	1.0000	1.0000	39	1.0000	7
RD100	1.0000	1.0002	20	1.0000	1.0032	61	1.0000	7
EIL101	1.0040	1.0053	23	1.0000	1.0013	64	1.0040	5
LIN105	1.0000	1.0000	24	1.0000	1.0000	35	1.0000	14
PR107	1.0000	1.0000	28	1.0000	1.0000	58	1.0000	9
PR124	1.0006	1.0033	41	1.0000	1.0016	81	1.0043	12
BIER127	1.0000	1.0310	55	1.0004	1.0051	56	1.0555	36
PR136	1.0000	1.0091	59	1.0001	1.0024	152	1.0128	13
PR144	1.0008	1.0008	62	1.0000	1.0001	105	1.0000	16
KROA150	1.0004	1.0006	68	1.0000	1.0001	179	1.0000	18
KROB150	1.0000	1.0030	68	1.0000	1.0042	107	1.0000	14
PR152	1.0000	1.0000	96	1.0000	1.0000	85	1.0047	18
U159	1.0000	1.0000	93	1.0000	1.0049	93	1.0260	19
RAT195	1.0012	1.0035	150	1.0012	1.0105	194	1.0000	37
D198	1.0049	1.0064	229	1.0049	1.0062	143	1.0060	60
KROA200	1.0000	1.0000	167	1.0072	1.0073	157	1.0000	30
KROB200	1.0041	1.0160	169	1.0035	1.0059	226	1.0000	36
TS225	1.0000	1.0012	250	1.0009	1.0034	364	1.0061	89
PR226	1.0000	1.0000	247	1.0000	1.0035	142	1.0000	26
GIL262	1.0000	1.0125	407	1.0128	1.0194	319	1.0503	115
PR264	1.0000	1.0022	408	1.0015	1.0034	323	1.0036	64
PR299	1.0015	1.0071	580	1.0035	1.0088	638	1.0223	90
LIN318	1.0000	1.0076	745	1.0010	1.0010	301	1.0459	207
RD400	1.0260	1.0309	1562	1.0105	1.0186	1533	1.0123	404
FL417	1.0112	1.0114	1890	1.0048	1.0048	461	1.0048	427
PR439	1.0129	1.0147	2208	1.0107	1.0148	867	1.0352	611
PCB442	1.0273	1.0508	2285	1.0075	1.0111	1167	1.0591	568
Avg.	1.0026	1.0062	336	1.0020	1.0039	230	1.0098	83
Nb. Optima	24	14		21	13		20	

Table 1: Computational results on the benchmark problems

5 Conclusion

In this paper, a genetic algorithm for solving the GTSP was presented and its competitiveness with the best heuristics known to date was empirically demonstrated on a set of benchmark problems. To obtain such results, the GA had to be hybridized with classical operations research techniques, to produce an ordering (i.e., a tour) with the vertices selected by the GA. A powerful mutation mechanism, based on local search, was also required.

Acknowledgments. Financial support for this work was provided by the Canadian Natural Sciences and Engineering Research Council (NSERC) and by the Quebec Fonds pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR). This support is gratefully acknowledged.

References

[1] Andresol R., M. Gendreau and J.-Y. Potvin, "A Hopfield-Tank Neural Network Model for the Generalized Traveling Salesman Problem", in *Meta-Heuristics - Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, S. Martello, I.H. Osman, C. Roucairol eds., Kluwer, 393–402, 1999.

[2] Baker J., "Reducing Bias and Inefficiency in the Selection Algorithm", in *Proceedings of the Second Int. Conf. on Genetic Algorithms*, Lawrence Erlbaum, 14–21, 1987

[3] Baker J., "Adaptive Selection Methods for Genetic Algorithms", in *Proceedings of an Int. Conf. on Genetic Algorithms and their Applications*, Lawrence Erlbaum, 101–111, 1985.

[4] Fischetti M., J.J.S. Gonzalez and P. Toth, "A Branch-and-Cut Algorithm for the Symmetric Generalized Traveling Salesman Problem", *Operations Research 45*, 378–394, 1997.

[5] Glover F. and M. Laguna, *Tabu Search*, Kluwer, 1997.

[6] Laporte G. and Y. Nobert, "Generalized Traveling Salesman Problem through n Sets of Nodes: An Integer Programming Approach", *INFOR 21*, 61–75, 1983.

[7] Laporte G., A. Asef-Vaziri and C. Sriskandarajah, "Some Applications of the Generalized Traveling Salesman Problem", *Journal of the Operational Research Society 47*, 1461–1467, 1996.

[8] Lawler E.L., J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys eds., *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, 1985.

[9] Lin S., "Computer Solutions of the Traveling Salesman Problem", *Bell System Technical Journal 44*, 2245–2269, 1965.

[10] Or I., "Traveling Salesman-type Combinatorial Problems and their Relation to the Logistics of Blood Banking", Ph.D. Thesis, Dept. of Industrial Enginnering and Management Sciences, Northwestern University, 1976.

[11] Potvin J.-Y., "Genetic Algorithms for the Traveling Salesman Problem", *Annals of Operations Research 63*, 339–370, 1996.

[12] Reinelt G., "TSPLIB - A Traveling Salesman Problem Library", *ORSA Journal on Computing 3*, 376–384, 1991.

[13] Renaud J. and F.F. Boctor, "An Efficient Composite Heuristic for the Symmetric Generalized Traveling Salesman Problem", *European Journal of Operational Research 108*, 571–584, 1998.

[14] Renaud J., F.F. Boctor and G. Laporte, "A Fast Composite Heuristic for the Symmetric Traveling Salesman Problem", *INFORMS Journal on Computing 8*, 134–143, 1996.

[15] Saksena J.P., "Mathematical Model of Scheduling Clients through Welfare Agencies", *CORS Journal 8*, 185–200, 1970.

[16] Semet F. and J. Renaud, "A Tabu Search Algorithm for the Symmetric Generalized Traveling Salesman Problem", Technical report CRT-99-19, Centre de recherche sur les transports, Université de Montréal, 1999.

[17] Srivastava S.S., S. Kumar, R.C. Garg and P. Sen, "Generalized Travelling Salesman Problem through n Sets of Nodes", *CORS Journal 7*, 97–101, 1969.

[18] Syswerda G., "Uniform Crossover in Genetic Algorithms", in *Proceedings of the Third Int. Conf. on Genetic Algorithms* Morgan Kaufmann, 2–9, 1989.

[19] Whitley D., "The GENITOR algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best", in *Proceedings of the Third Int. Conf. on Genetic Algorithms*, Morgan Kaufmann, 116–121, 1989

Acceptance Driven Local Search and Evolutionary Algorithms

Eric Poupaert and Yves Deville*
Department of Computing Science and Engineering
Université catholique de Louvain
Place Sainte Barbe 2, B-1348 Louvain-la-Neuve, Belgium
Email: {ep.yde}@info.ucl.ac.be
Tel: +32 10 473150

Abstract

Local Search (LS) and Evolutionary Algorithms (EA) are probabilistic search algorithms, widely used in global optimization, where selection is important as it drives the search. In this paper, we introduce *acceptance*, a metric measuring the selective pressure in LS and EA, that is the trade-off between exploration and exploitation. Informally, acceptance is the proportion of accepted non-improving transitions in a selection.

We propose a new LS algorithm, SAAD, based on acceptance schedule (a schedule for the selective pressure). In EA, two new selection rules based on the Metropolis criterion are introduced. They allow two new EA (2MT and RT) based on acceptance schedule. They demonstrate a possible way of merging LS and EA technologies. Benchmarks show that the developed algorithms are more performant than standard SA and EA algorithms, and that SAAD is as efficient as the best SA algorithms while 2MT and RT are complementary to Evolution Strategies.

1 Introduction

Local Search (LS) and Evolutionary Algorithms (EA) are probabilistic search algorithms widely used in global optimization. Such problems can be formalized as a set of solutions (called search space) and a function evaluating the solutions (called score, energy or fitness). The aim of global optimization is to find a solution such that no other solution is better. LS is based on the concept of neighborhood; its principle is to improve iteratively a current solution by generating and selecting neighbor solutions. The principle of EA is to model the evolution of a population of individuals through recombination, mutation and selection.

*This research is partially supported by the *actions de recherche concertée* ARC/95/00-187.

Selection is an important part of both LS and EA: it drives the search toward promising zones of the search space. Selection is subject to an important trade-off: it either favors the *exploration* of the search space or the *exploitation* of the neighborhood (or population). This trade-off is usually expressed using the informal term of *selective pressure*: high pressure implying exploitation and low pressure exploration. Measuring selective pressure is an important trend in EA. Takeover time, for example, is the metric used in (Bäck, 1994).

A contribution of this paper is to provide a metric measuring the selective pressure appropriate for both LS and EA. We will therefore introduce the notion of *acceptance*, the proportion of accepted non-improving transitions in a selection.

The best-known EA are: Genetic Algorithms (GA, Holland, 1975), Evolution Strategies (ES, Bäck, 1996) and Evolutionary Programming (EP, Fogel, 1992).

In the vast majority of EA, selection does not vary during the search. However, a varying selection parameter is used in an example in (Davis, 1991), and a Boltzmann Tournament Selection aiming at a niching mechanism is described in (Goldberg, 1990).

Simulated Annealing (SA, Kirkpatrick et al., 1983) is a LS algorithm originating from a simulation of the thermodynamics of gas. In SA, selection is performed using the Metropolis criterion (Metropolis et al., 1955) which has a parameter called temperature. The principle of SA is to enforce a temperature schedule reducing temperature progressively during optimization. This reduction of temperature permits to achieve a convergence to the global optimum. The temperature schedule is critical to the success of SA.

The best-known temperature schedule SAGEO is the geometric one (Kirkpatrick et al., 1983). Another important one is SAPOLY (van Laarhoven, 1988) where temperature reduction is performed using a feedback mechanism based on the concept of quasi-equilibrium.

It has a polynomial time complexity. In addition to an analog feedback, SAEF innovates by introducing a variable chain length based on an approximate measure of equilibrium. It is considered as one of the best SA known of to date (Aarts and Lenstra, 1997).

In SA, temperature has a direct impact on the selective pressure which initially is low and increases with time. The progressive increase of selective pressure is the core of of SA. In LS and in EA, although selection may vary during the search, the resulting selective pressures can only de deduced during the execution. In existing algorithms, selection is not adapted according to a given schedule for the selective pressure.

A contribution of this paper is the definition of *acceptance schedule* (a schedule for the selective pressure) and an associated algorithm (ESTIMATE_PARAMETER) computing the successive values of a selection parameter (temperature) in order to achieve an acceptance schedule. Since acceptance is appropriate for both LS and EA, it *makes possible the merging of SA and EA technologies.*

The other contributions of this paper are now described. *We designed and implemented a new local search algorithm*, SAAD, based on acceptance schedule. Its temporal complexity can be *a priori* computed and is $O(v \log v)$ (where v is the average size of a neighborhood). Benchmarks have shown that SAAD is more performant than standard SA techniques, and is as efficient as the best SA algorithms. *We defined two selection rules*, a relaxed 2-Tournament and a relaxed truncation, applicable in EA. These rules introduce the Metropolis criterion on populations and allow for adaptable acceptance. They respect the design guidelines expressed in (Bäck, 1994). *We designed and implemented new evolutionary algorithms*, 2MT and RT, based on acceptance schedule and implementing the two selection rules. Benchmarks have shown they are more performant than standard EA, and are complementary to ES.

The paper is structured as follows. In Section 2, acceptance driven SA is presented; acceptance and acceptance schedule are defined and algorithm SAAD is described. Section 3 presents acceptance driven EA; two selection rules are proposed and algorithms EAAD is described. Experimental results are analyzed in Section 4.

2 Acceptance Driven SA

2.1 Definition of Acceptance

A transition occur when the current solution is replaced by one of its neighbors. Transitions that improve the current solutions are natural since they contribute to both the exploration of the search space and the exploitation of the neighborhood. Non improving transitions go in the direction of exploration but to the detriment of exploitation. Selective pressure is related to the probability that these transitions occur.

We propose a new metric of selection pressure, called *acceptance*. Intuitively, it is the proportion of non-improving transitions that are accepted. Moreover, it is a global measure of the solution space and is not relative to a specific current solution.

Definition: Given a local search algorithm LS using a selection rule SELECT, a neighbor function NEIGHBOR and an energy function ENERGY,

$$\begin{aligned} \textit{acceptance} = \mathcal{P}(&\textsc{select}(S, S', t) = S' \mid \\ &\textsc{energy}(S') > \textsc{energy}(S) \ \&\ S' = \textsc{neighbor}(S)) \end{aligned}$$

where S and S' are solutions and t a parameter of the selection. The upper bound 1 of acceptance implies that non improving transitions are always accepted; and the lower bound 0 that they are never accepted. Acceptance is relative to a selection rule SELECT and its parameter t. As t may vary during the search, such as in SA, acceptance may also vary.

2.2 Local Search Driven by Acceptance

```
1   S := INITIAL_SOLUTION
2   s := 0
3   while CONTINUE do
4       χ := TARGET_ACCEPTANCE(s)
5       t := ESTIMATE_PARAMETER(χ)
6       repeat L times
7           S' := NEIGHBOR (S)
8           S := SELECT (S,S',t)
        end
9       s := s + 1
    end
10  return S
```

Algorithm 1: Acceptance driven EA : SAAD

Temperature reduction is the core of SA. In Algorithm 1, we propose a new LS algorithm, called Simulated Annealing Driven by Acceptance (SAAD), based on SA with an acceptance schedule. To this end, a parameter χ and an index s are used.

Selection is performed in SA using the Metropolis criterion (Algorithm 2). Its parameter t controls the selective pressure. Hence

$$\textsc{Select}(S, S', t) \Leftrightarrow \textsc{Metropolis}(S, S', t)$$

The relationship between acceptance and the selection parameter (temperature in SA) is performed by the ESTIMATE_PARAMETER function specified hereafter. The successive iterations where the parameter is kept constant is called a *chain*.

function $S'' = \text{METROPOLIS}(S,S',t)$
begin
$\Delta := \text{ENERGY}(S') - \text{ENERGY}(S)$
$p := \min(1, \exp(-\Delta/t))$
if $\text{RANDOM}(0,1) < p$ **then** $S'' := S'$ **else** $S'' := S$
end

Algorithm 2: Metropolis criterion

function $t = \text{ESTIMATE\_PARAMETER}\ (\chi)$
Pre: $\chi \in [0,1]$
Post: $t \geq 0$ such that the expected acceptance is equal to χ for a chain using t as the value of the parameter of SELECT .

An *acceptance schedule* is a method to determine, for each moment of the search, an acceptance we would like to enforce (called *target acceptance*). This is the role of TARGET_ACCEPTANCE.

function $\chi = \text{TARGET\_ACCEPTANCE}(s)$
Pre: $s \geq 0$ is the index of a chain
Post: $\chi \in [0,1]$ is the target acceptance for the chain of index s

2.3 Acceptance Schedule

The initial value χ_0 of acceptance, $\chi_0 = 1$, is its upper bound. It leads to a complete coverage of the search space.

The next step is to determine the decrease of acceptance. A well-known heuristic used in simulated annealing states that "the number of (accepted) transitions must be constant for each chain". This implies that chains of lower temperature (and thus lower acceptance) must be longer. We have modified this heuristic to fit our framework: "the number of (accepted) transitions per unit of acceptance is constant", that is $\chi(s).L/(\chi(s+1)-\chi(s))$ (where s is the chain index) is constant. This implies that more time must be spent for lower acceptance. This leads to a differential equation whose solutions are: $\chi(s) = \alpha \cdot \exp(-\beta \cdot s)$. We have also $\alpha = \chi(0) = 1$. As $\chi(s)$ forms a geometric sequence, it can be expressed in term of half-life (denoted $s_{1/2}$): the number of chains such that the acceptance is divided by two (i.e. $\chi(s + s_{1/2}) = \chi(s)/2$, where $s_{1/2}$ is an input parameter of the algorithm). We obtain finally:

$$\text{TARGET\_ACCEPTANCE}(s) = \chi := (0.5)^{s/s_{1/2}} \qquad (1)$$

Since we want acceptance to decrease with time, we have $s_{1/2} > 0$. A higher value of $s_{1/2}$ corresponds to a slower decrease of χ.

The stopping condition is traditionally seen as being part of a schedule. The criterion usually used in local search is to stop when no further improvement of the current solution is to be expected. In our implementation, we have chosen to stop when the expected number of non improving transitions during *stop* (an input parameter of the algorithm) consecutive chains is below 1/2:

$$\text{CONTINUE} \Leftrightarrow \mathit{stop} \cdot L \cdot \chi \geq 1/2 \qquad (2)$$

In practice, L is set to 3 times the size of the neighborhood, *stop* between 5 and 10 and $s_{1/2}$ is set according to the time available for optimization.

2.4 Estimation of the Selection Parameter

At the beginning of each chain, the parameter t has to be estimated such that the expected acceptance over this chain is equal to the target acceptance χ. This estimation uses a feedback mechanism. Given a transition $S \to S'$ and the value of t, the probability that this transition is accepted can be computed from the SELECT function. Likewise, the acceptance over a chain can be computed *a posteriori* knowing the transitions proposed by NEIGHBOR.

Let $acc(H,t)$ be a measure of the acceptance, called *acceptance function*, over a chain where H is the set of the (proposed) transitions. We have:

$$acc(H,t) = \mathcal{P}(\text{SELECT}(S,S',t) = S' \mid \text{ENERGY}(S') > \text{ENERGY}(S)\ \&\ (S \to S') \in H)$$

Estimating t_i for the chain of index i is therefore equivalent to find t_i such that $acc(H_i,t_i) = \chi_i$.

Unfortunately, H_i is not known *a priori*; especially since H_i results from a stochastic process involving the value of t_i. A way to solve this problem is to suppose that $acc(H_i,t_i) \simeq acc(H_{i-1},t_i)$ because H_{i-1} will be known when t_i will have to be computed. This hypothesis is realistic when t_i and t_{i-1} are not too distant and when H_{i-1} is large enough to be a good representative of the transitions that *could* have been proposed during this chain. From a statistical point of view, it is the case when L is of the same order as the size of the neighborhood.

Estimating t_i is thus equivalent to solving $acc(H_{i-1},t_i) = \chi_i$, where H_{i-1} and χ_i are known.

Given that selection is based on the Metropolis criterion, the acceptance function can easily be computed from a set H of non improving transitions.

$$acc(H,t) = (1/n) \cdot \textstyle\sum_{j=0}^{n-1} \exp(-\Delta_j/t) \qquad (3)$$

where $\Delta_j > 0$ is the energy difference of the jth transition of H and n the size of H.

In order to implement the ESTIMATE_PARAMETER function, the acceptance function must be inverted; this is possible as $acc(H,t)$ is monotonous relatively to t. To this end, a Newton-Raphson (N-R) method can be used. For numerical stability reasons, we preferred to use a variation of N-R, working on the logarithm of the parameter. The principle is to find a root of a

function by forming a sequence of better and better estimates of the root. Suppose that we want to find the root of the equation $f(x) = 0$ and that we have an initial estimate x_0 of a solution, the (variant of) N-R sequence is:

$$x_{k+1} = x_k \cdot \exp\left(\frac{-f(x_k)}{f'(x_k)/x_k}\right) \quad (4)$$

where f' is the derivative of f. If the sequence converges, its limit is a solution of the equation.

In our case, the equation $acc(H,t) - \chi = 0$ must be solved. We therefore have:

$$t_{k+1} = t_k \cdot \exp\left(\frac{n \cdot \chi - \sum_{j=0}^{n-1} \exp(-\Delta_j/t_k)}{\sum_{j=0}^{n-1} (\Delta_j/t_k) \cdot \exp(-\Delta_j/t_k)}\right) \quad (5)$$

It is evidenced by experiments that if the temperature of the previous chain is used as initial estimate of the one of the next chain, a single step of N-R leads to an appropriate precision.

Without initial estimate, t can be approximated as

$$t \approx -\overline{\Delta}/\ln(\chi) \quad (6)$$

where $\overline{\Delta}$ is the average of the various Δ_j. This estimation is only accurate when $\chi > 0.9$.

The proposed algorithm must now be accommodated to maintain a set of transitions:

```
1.1   H := ∅
5.1   t := ESTIMATE_PARAMETER(H,χ)
5.2   H := ∅
8.1   if ENERGY(S') > ENERGY(S)
8.2       then H := H ∪ {S → S'}
8.3   S := SELECT (S,S',t)
```

In practice, each time a non improving transition is proposed, the contribution of the corresponding Δ to the sums of Eqs. 5 and 6 are accumulated. Therefore, H is not stored as an explicit set of transitions but as quadruplet (H_0, H_1, H_2, H_3) where:

H_0 = n (number of transitions, see Eqs. 5 and 6)
H_1 = sum of $\exp(-\Delta_j/t)$ (see Eq. 5)
H_2 = sum of $(\Delta_j/t) * \exp(-\Delta_j/t)$ (see Eq. 5)
H_3 = sum of Δ_j/t (see Eq. 6)

These variables can be updated by simple instructions replacing Instruction 8.2. as the transitions themselves are now useless. An implementation of the ESTIMATE_PARAMETER function is given in Algorithm 3.

2.5 Complexity

Let ep be the complexity of ESTIMATE_PARAMETER, ng of NEIGHBOR, sel of SELECT, en of ENERGY and s_{max} the total number of chains. The complexity of the algorithm SAAD is:

$$O(s_{max} \cdot ep + s_{max} \cdot L \cdot (ng + sel))$$

```
function t = ESTIMATE_PARAMETER(H,χ,t_old)
    begin
1       if H_0 = 0 then t := ∞
2       else if χ > 0.9 then t := −H_3/(H_0 · ln(χ))
3       else t := t_old · exp((H_0 · χ − H_1)/H_2)
    end
```

Algorithm 3: ESTIMATE_PARAMETER

The value of s_{max} can be derived from the acceptance schedule (Eq. 1) and the stopping condition (Eq. 2):

$$s_{max} = s_{1/2} \cdot (1 + \log_2(L \cdot stop))$$

Moreover $ep = O(1)$. The $stop$ parameter is problem independent and can thus be seen as a constant (fixed between 5 and 10 in our implementation). Using Metropolis, we have $sel = O(en)$. For most problems $ng = O(1)$ (generation and choice of a neighbor), and $en = O(1)$ as the energy can be computed incrementally within the NEIGHBOR function. As already justified, we fixed L to $O(v)$, where v is the average size of a neighborhood. The complexity of Algorithm SAAD becomes finally

$$O(s_{1/2} \cdot v \cdot \log(v)) \quad (7)$$

The space complexity of SAAD is $O(L \cdot \text{size}(S))$ but is reduced to $O(1)$ using the proposed implementation of the acceptance function.

It is noteworthy that the temporal complexity of SAAD, which also is the total number of generated neighbors ($L \cdot s_{max}$), can be computed *a priori*. This is usually not the case for classical SA algorithms, where a temporal complexity in often difficult to derive. In (van Laarhoven and Aarts, 1987), it is shown that for specific decrement rules and stop criteria, the total number of generated neighbors is $O(v \log(q))$, where q is the size of the set of configurations (usually exponential).

3 Acceptance Driven EA

In LS, the current state is the so-called current solution; in EA, it is a population. The evolution of this current population is performed by generating a second population (called offspring of the first) through the MUTATE and RECOMBINE functions, and, selecting individuals within these populations.

Our acceptance driven EA, called EAAD, is given in Algorithm 4. It is obtained by adapting SAAD (Alg. 1) to fit in a standard EA scheme, where P denotes the current population and contains n individuals, and P' a population offspring from P and contains m individuals. The role of SELECT is reflected in the following specification:

```
1   P := array: [1..n] → Individual
2   for each i ∈ [1..n]: P[i] := RANDOM_INDIVIDUAL
3   s := 0
4   while CONTINUE do
5       χ := TARGET_ACCEPTANCE(s)
6       t := ESTIMATE_PARAMETER(χ)
7       repeat L/m times
8           P' := MUTATE(RECOMBINE(P))
9           P := SELECT (P, P', t)
10      s := s + 1
11  return BEST_OF (P)
```

Algorithm 4: Acceptance driven EA (EA$_{AD}$)

function $P'' =$ SELECT (P, P', t)
Post: $P'' \subseteq P \cup P'$ and $\#P'' = \#P$
Note: t has an impact on the selective pressure

In EA$_{AD}$, the length of the chains is L/m, where m is the size of population P'. Hence, in terms of number of individuals to be evaluated, the length of the chains is L, as in SA$_{AD}$.

The CONTINUE and TARGET_ACCEPTANCE functions can be implemented as in SA$_{AD}$. To complete the EA$_{AD}$ algorithm, the ESTIMATE_PARAMETER and the SELECT functions have to be implemented.

3.1 Acceptance within Populations

As selections are performed on populations, the definition of acceptance given in Section 2 must be generalized. It is convenient to introduce a reference selection rule (SELECT_REF) stating which transitions should be accepted and which ones should be rejected using an exploitation oriented view. It thus has no third parameter.

Let $P'' =$ SELECT(P, P', t) (or $P'' =$ SELECT_REF(P, P')), and let $S \to S'$ be a transition with $(S, S') \in P \times P'$. This transition is *accepted* if $S \notin P''$ (S was in P but no longer in P''), and $S' \in P''$ (S' is selected in P'' from P'). This transition is *rejected* if $S \in P''$ (S was in P and is kept in P'') and $S' \notin P''$ (S' was a potentially new candidate from P', but is not selected in P''). The other possible transitions are meaningless and are thus neither accepted nor rejected.

If we take a 2-Tournament as SELECT_REF, a transition $S \to S'$ is rejected if it is non improving. In this case, each transition is evaluated separately.

On the other hand, if Truncation is used as SELECT_REF, the transitions are evaluated globally. A transition $S \to S'$ is rejected if S is in the set of the n best individuals of $P \cup P'$ and S' is not in this set (n is the size of P).

Definition: Given an evolutionary algorithm EA$_{AD}$ and a function SELECT_REF, the *acceptance* of EA$_{AD}$ is the expected proportion of transitions generated using RECOMBINE and MUTATE and rejected by SELECT_REF that are accepted by SELECT. Formally:

$$\begin{aligned} \mathit{acceptance} = \mathcal{P}(&S \notin P'' \ \&\ S' \in P'' \mid \\ &P' = \text{MUTATE}(\text{RECOMBINE}(P)) \\ &\&\ P'' = \text{SELECT}(P, P', t) \ \&\ P''' = \text{SELECT\_REF}(P, P') \\ &\&\ (S, S') \in P \times P' \ \&\ S \in P''' \ \&\ S' \notin P''' \ ) \end{aligned}$$

When P and P' are singletons, this definition is equivalent to the acceptance defined for SA$_{AD}$.

In LS$_{AD}$, the selection parameter t was estimated by inverting an acceptance function $acc(H, t)$ measuring the acceptance for a chain H. As H is now a set of transitions on *populations*, the acceptance function must also be generalized. Let $acc'(H', t)$ be a measure of the acceptance over a chain where H' is a set of transitions between populations and t the parameter of SELECT. We call it *acceptance function on populations*, and it is defined as follows :

$$\begin{aligned} acc'(H', t) = \mathcal{P}(&S \notin P'' \ \&\ S' \in P'' \mid (P \to P') \in H' \\ &\&\ P'' = \text{SELECT}(P, P', t) \ \&\ P''' = \text{SELECT\_REF}(P, P') \\ &\&\ (S, S') \in P \times P' \ \&\ S \in P''' \ \&\ S' \notin P''' \ ) \end{aligned}$$

Handling a set of transitions between populations would be too complex. It is therefore convenient to transform a set of transitions between populations into a set containing the relevant transitions (between individuals), that is the transitions involved in the conditional part of the acc' function. Formally, given $H' = \{P \to P'\}$, the *set of relevant transitions* (between individuals) is the set

$$\begin{aligned} H = \{S \to S' \mid &(P \to P') \in H' \\ &\&\ P''' = \text{SELECT\_REF}(P, P') \\ &\&\ (S, S') \in P \times P' \ \&\ S \in P''' \ \&\ S' \notin P''' \ ) \end{aligned}$$

In Algorithm EA$_{AD}$, the acceptance is varying from one to (nearly) zero. An acceptance of one implies a selection equivalent to SELECT_REF. An acceptance of zero implies a random selection out of the union of the populations. Therefore, SELECT can be seen as a variable relaxation of the reference selection rule.

We are now in position to present two particular SELECT functions, and their associated ESTIMATE_PARAMETER. In both selections, a relaxation is introduced using the Metropolis criterion.

3.2 2-Metropolis-Tournament

In a 2-tournament, pairs of individuals are formed and for each pair, the best individual of the two is selected. We have developed a relaxed 2-tournament using the Metropolis criterion. The idea is use the Metropolis criterion on each pair to elect the selected individuals. When the winner is elected using a non-deterministic criterion, fairness imposes that each individual enters

in a constant number of trials. Therefore, we have decided to make pairs without replacement. Moreover, since the Metropolis criterion is asymmetrical (favors new solutions), we have chosen to make asymmetrical pairs: the first individual always comes from the current population and the second from its offspring. In this case, unless a lazy approach is used, it is useful to impose that the populations P and P' have the same size n.

```
    function P'' = SELECT (P, P', t)
    Pre: #P = #P'
    begin
1       n := SIZE_OF (P)
2       P'' := array: [1..n] → Individual
3       P' := PERMUTE (P')
4       for each i ∈ [1..n] :
5           P''[i] := METROPOLIS (P[i], P'[i], t)
    end
```

Algorithm 5: 2-Metropolis-tournament

A 2-Metropolis-tournament is implemented in Alg. 5. In line 3, P' is rearranged in a random order so that $(P[i], P'[i])$ form random asymmetrical pairs without replacement. In line 5, the Metropolis criterion (see Alg. 2) is used on each pair to elect a winner which is added to P''.

This selection rule could be used in any EA extended with temperature or acceptance schedule.

3.3 2MT: EAAD with 2-M.-tournament

One could show that in 2-Metropolis-tournament, $acc'(H', t) = acc(H, t)$, where H is the set of relevant transitions from H' (acc is the acceptance function defined in Section 2). Intuitively, individuals of P' are selected independently of each other (this is also true for P). Therefore, the transitions between individuals are also accepted independently.

In the context of 2MT, the set of relevant transitions becomes $H = \{S \to S' \mid P \to P' \in H' \ \& \ (S, S') \in P \times P' \ \& \ \text{ENERGY}(S') > \text{ENERGY}(S')\}$. Therefore, the parameter t can be estimated using ESTIMATE_PARAMETER as implemented in SAAD.

In practice, a sample (of size L) of H can be used in place of H (of size $L \cdot n$). Such a sample can be formed easily by random non-improving transitions from P to P'. This function is denoted SAMPLE_TRANSITIONS.

The proposed EAAD algorithm must now be accommodated to maintain this set of transitions.

```
3b   H = ∅
6.1  t := ESTIMATE_PARAMETER(H, χ, t)
6.2  H = ∅
9.1  H = H ∪ SAMPLE_TRANSITIONS(P, P')
9.2  P := SELECT (P, P', t)
```

One could easily show that 2MT has the same temporal complexity than SAAD, that is $O(s_{1/2} \cdot v \cdot \log(v))$, where v is the average size of a neighborhood.

3.4 Relaxed Truncation

In this section, we design a new selection rule based on a relaxed truncation using the Metropolis criterion. What is needed is a relaxed sorting algorithm; the quality of sorting being subject to a parameter t. Different schemes could be used: in a first one, the standard key comparator is replaced by a stochastic one; in another one, the keys receive a stochastic amount of perturbation. With the first scheme, the acceptance function depends on the chosen sorting algorithm. Therefore, the second scheme is preferred.

The Metropolis criterion can be viewed as a way to sort two solutions. Imagine the situation where S and S' are solutions, x and x' their respective energy, and that S is better than S' (i.e. $\Delta = x' - x > 0$). In the Metropolis algorithm (Alg. 2), we see that S' wins (becomes first) when

$$r < \exp(-\Delta/t) \quad \Leftrightarrow \quad x' < x - t \cdot \ln(r)$$

where r is a random value with a uniform distribution over $[0..1]$. When $t = 0$, a non improving transition can never be accepted. When t increases, an increasing value is added to x and therefore the probability that S' wins increases. The Metropolis criterion is asymmetrical: a penalty is added to x only.

This scheme is extended to populations in Alg. 6. Every individual of P receives a penalty of the form $-t\ln(r)$ (with a different r for each individual). Individuals of P' have no penalty. The mapping v contains the energy plus penalty of every individual. The individuals of $P \cup P'$ are sorted according to this mapping. The set P'' is composed of the first individuals of the sorted union such that P'' and P have the same size.

```
    function P'' = SELECT (P, P', t)
    begin
1       v := map: Individual → real
2       for each p ∈ P :
3           v[p] := ENERGY(p) − t ln(RANDOM(0, 1))
4       for each p ∈ P' : v[p] := ENERGY(p)
5       P'' := P ∪ P'
6       SORT (P'', v)
7       TRUNCATE(P'', SIZE_OF(P))
    end
```

Algorithm 6: Relaxed truncation

The relaxed truncation selection rule can be used in any EA extended with temperature or acceptance schedule.

3.5 RT: EAAD with Relaxed Truncation

Relaxed truncation accepts individuals using a global approach. Therefore, transitions between individuals are not accepted independently. Hence the relation $acc' = acc$ does not hold here. However, strong experimental evidences show that

$$acc'(H', t) = \frac{n}{n+m} \cdot acc(H, t)^k \qquad (8)$$

where k is a constant and H is the set of relevant transitions from H' (with SELECT_REF implemented by Truncation). The term $n/(n+m)$ is the upper bound of acceptance for SELECT (when $t = \infty$). The constant k appeared to be independent of the statistical distribution of the transitions and can be easily computed by simulation. In practice, $k = 0.5$ when the population is large ($(n, m) = (7, 50)$) and decreasing slowly toward 1 for smaller populations.

To complete the algorithm, a sample of the set of relevant transitions (e.g. SELECT(P,P',0) in place of SELECT_REF) should also be computed here. This sampling can be achieved as in 2MT. The resulting complexity of SELECT and small SAMPLE_TRANSITIONS is $O((n+m) \cdot \log(n+m))$. Moreover, since ESTIMATE_PARAMETER inverts $acc(H, t)$ and not $acc'(H', t)$, its argument must also be adapted using Eq. 8.

6.1.1 $\chi' = (\chi/(n/n+m)) **(1/k)$
6.1.2 $t :=$ ESTIMATE_PARAMETER(H, χ', t)

As the ratio n/m is a constant (generally fixed to $1/7$), one could easily show that the temporal complexity of RT is $O(s_{1/2} \cdot v \cdot \log(v) \cdot \log(n))$.

4 Experimental Results

The aim of this section is to compare experimentally the proposed algorithms to relevant EA and SA algorithms. For space reasons, only the most relevant experiments are reported. Our analysis is however based on the whole set of experiments.

SAAD is first experimentally compared to classical SA algorithms (SAGEO, SAPOLY and SAEF). For each of these algorithms every parameter is set according to their respective authors recommendations. When a range is proposed, the best values in that range are used. Benchmarks are performed on three TSP instances from the TSPLIB (Reinelt, 1991) (berlin52, ch130 and a280) and on F6. F6 (Davis, 1991) is a moderately multimodal function of low dimensionality ($k = 3$). Table 1 summarizes these experiments. s is the total number of iterations, ϵ is the relative error of the energy of the final solution (compared to the known optimal energy), $s_{1/2}$ is the number of iterations giving a success rate equivalent to 50%. Each line is the result of at least 100 runs.

Problem	Algorithm	Parameters	Results
F6	SAEF		$n_{1/2} = 565\ 296$
F6	SAAD		$n_{1/2} = 429\ 286$
berlin52	SAGEO		$n_{1/2} = 368\ 211$
berlin52	SAPOLY		$n_{1/2} = 223\ 949$
berlin52	SAEF		$n_{1/2} = 150\ 156$
berlin52	SAAD		$n_{1/2} = 163\ 838$
ch130	SAEF	$s = 900\ 000$	$\epsilon = 2.83\%$
ch130	SAAD	$s = 900\ 000$	$\epsilon = 2.78\%$
a280	SAEF	$s = 5\ 250\ 000$	$\epsilon = 3.42\%$
a280	SAAD	$s = 5\ 250\ 000$	$\epsilon = 3.20\%$

Table 1: Comparaison of SA algorithms

	F6		F9		
Algorithms	p	Err.	$\bar{e}$	Std e	Err.
SAAD	18%	±2.4%	265	79	±31
2MT (20,20)	90%	±5.9%	58	13	±5
RT (7,50)	98%	±2.7%	97	22	±9
SHC	6%	±4.7%	260	34	±13
2T (20,20)	26%	±8.6%	71	13	±5
T (7,50)	15%	±7.0%	88	17	±7
ES (7,50)	23%	±8.2%	32	11	±4
ES (30,200)	50%	±9.8%	183	76	±30

Table 2: Results for F6 ($k = 3$) and F9 ($k = 30$)

This study concludes that SAAD and SAEF outperform SAGEO and SAPOLY. SAAD and SAEF exhibit similar performances on the TSP instances with a slight advantage for SAEF on the largest instance. SAAD shows better performances on F6.

Since selection is independent from the problem and from the mutation / recombination operators (Bäck, 1994), our algorithms (SAAD, 2MT and RT) are compared with other EA, identical in every aspects, but using standard selection rules. These algorithms do not use an acceptance schedule nor another form of adaptative selection. They are denoted by SHC (classical Stochastic Hill Climbing), 2T (EA with select = 2-Tournament), T (EA with select = Truncation as in ES-(n+m)). It is also interesting to compare the proposed algorithms to Evolution Strategies (ES) since they use a different (but complementary) approach.

These seven algorithms are compared on two function optimization problems (F6 and F9). F9 is the well-known problem due to Rastrigin generalized as in (Yao and Liu, 1997). It has a high dimensionality ($k = 30$) and is highly multimodal; it is considered difficult for most optimization methods.

For ES, every parameter or choice is made in conformance with the recommendations found in (Bäck, 1996): k standard deviations are used; the solutions are recombined either using a discrete or a panmictic discrete operator (depending on what make most sense for each problem); the standard deviations are recombined using a panmictic intermediate operator; the

selection is either ES-(n+m) or ES-(n,m) (whichever gives the best results). For the other algorithms: a log-uniform mutation is used (a value $s \cdot 10^r$ is added to each coordinate, where s is a random sign +1 or -1 and r is a random uniformly distributed real value over [1,-4]); the recombination of the solutions are the same as for ES. For SAAD, 2MT and RT, $s_{1/2} = 10$, $L = 1000$ and $stop = 10$. This lead to a total of 144000 generated (and evaluated) individuals. In order to have a fair comparison, all the algorithms are terminated when this number of generated individuals is reached.

All the compared algorithms have been implemented in Java. Source code is available upon request to the first author.

The optimal energies of F6 and F9 is 0. For F6, we measure the proportion p of 100 (independent) runs of the algorithms that lead to the optimal solution (i.e. with a maximal error of 1e-4). For F9, the optimal solution were never reached during the experiments, therefore, we measure the mean energy $\bar{e}$ of the best individual of the final population on 25 runs. Confidence intervals of 95% for p and $\bar{e}$ are also computed. The results are summarized in Table 2.

On F6, 2MT and RT are the best performers by far with ES being in third place. On F9, ES is best and 2MT is second.

On these experiments, the population based algorithms (2MT, RT, 2T, T and ES) show generally better performance than the corresponding solution based ones (SAAD and SHC). The algorithms based on an adaptative selection (SAAD, 2MT and RT) or on an adaptative mutation (ES) perform generally better than their non adaptative counterparts (SHC, 2T and T). The adaptative selections perform particularly well on F6 and the adaptative mutations on F9. Both approaches are complementary and could be combined.

5 Conclusion

In this paper we designed and experimented three new local search and evolutionary algorithms (SAAD, 2MT and RT). They are based on acceptance schedule, an original a schedule for the selective pressure. The successive values of a selection parameter are computed in order to achieve an acceptance schedule. This was impossible with traditional LS and EA algorithms. We thus demonstrate a possible way of merging SA and EA technologies.

Our notion of *acceptance* is a measure of compromise between exploitation and exploration. It takes into account the selection and the generation of neighbors, and is appropriate for both LS and EA.

Adaptable acceptance has been introduced in EA through two new selection rules, introducing the Metropolis criterion on population.

The temporal complexity of the algorithms has been analyzed. They can be computed *a priori*, hence the execution time can be predicted.

Experiments show that the developed algorithms are more performant than standard SA and EA algorithms, and that SAAD is as efficient as the best SA algorithm while 2MT and RT are complementary to Evolution Strategies.

This research aims at developing adaptability in LS and in EA. Acceptance driven algorithms should be seen as a possible way to introduce adaptability. Adaptative mutation, such as in (Bäck, 1996), is a complementary approach. Further work includes the combination of acceptance schedule with adaptative mutation, and a characterization of classes of problems where acceptance schedule can be fruitful.

References

E. Aarts and J. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, 1997.

T. Bäck. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Proc. of the First IEEE Conf. on Evolutionary Computation*, pages 57–62. IEEE Press, 1994.

T. Bäck. *Evolutionary Algorithms in Theory and Practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, 1996.

L. Davis, editor. *Handbook of Genetic Algorithms*, 1991. Van Nostrand Reinhold.

D. Fogel. *System identification through simulated evolution: a machine learning approach to modeling*. Ginn Press, 1992.

D. Goldberg. A note on boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. *Complex Systems*, 4:445–460, 1990.

J. Holland. *Adaptation in Natural and Artificial Systems*. Ann Arbor: The University of Michigan Press, 1975.

S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1955.

G. Reinelt. Tsplib - a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

P. van Laarhoven. *Theoretical and Computational aspects of Simulated Annealing*. PhD thesis, Erasmus University Rotterdam, 1988.

P. van Laarhoven and E. Aarts. *Simulated Annealing: Theory and Applications*. D. Reidel Pub. Company, 1987.

X. Yao and Y. Liu. Fast evolution strategies. *Control and Cybernetics*, 26(3):467–496, 1997.