Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computers & Operations Research 33 (2006) 1214–1225
computers & operations research
www.elsevier.com/locate/cor

MA|PM: memetic algorithms with population management

Kenneth Sörensen[a,*], Marc Sevaux[b]

[a] *University of Antwerp, Faculty of Applied Economics, Prinsstraat 13, B-2000 Antwerp, Belgium*
[b] *University of Valenciennes, CNRS, UMR 8530, LAMIH-SP, Le Mont Houy-Bat Jonas 2, F-59313 Valenciennes cedex 9, France*

Available online 26 November 2004

Abstract

A new metaheuristic for (combinatorial) optimization is presented: memetic algorithms with population management or MA|PM. An MA|PM is a memetic algorithm, that combines local search and crossover operators, but its main distinguishing feature is the use of distance measures for *population management*. Population management *strategies* can be developed to dynamically control the diversity of a small population of high-quality individuals, thereby avoiding slow or premature convergence, and achieve excellent performance on hard combinatorial optimization problems. The new algorithm is tested on two problems: the multidimensional knapsack problem and the weighted tardiness single-machine scheduling problem. On both problems, population management is shown to be able to improve the performance of a similar memetic algorithm without population management.

Keywords: Memetic algorithm; Population management; Distance measures; Diversification

1. Introduction

"Classical" genetic algorithms (GA)—pioneered by, among others, Holland [1] and Goldberg [2]—have been shown experimentally to perform rather poorly on many combinatorial optimization problems. The performance of many GA implementations is characterized by large memory requirements, large running times and poor performance in terms of solution quality.

* Corresponding author. Tel.: +32 3 220 40 49.
E-mail address: kenneth.sorensen@ua.ac.be (K. Sörensen).

0305-0548/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2004.09.011

Commonly referred to as evolutionary algorithms (EA), many variants of genetic algorithms have been proposed to improve their performance. Perhaps the most successful in combinatorial optimization are *memetic algorithms* [3], also called *hybrid genetic algorithms* or *genetic local search*. These algorithms apply *local search operators* to improve the quality of individual solutions in the population.

Nevertheless, even memetic algorithms may still fall victim to either slow or premature convergence. Many researchers agree that the quality of a metaheuristic optimization approach is largely a result of the interplay between *intensification* and *diversification* strategies (see e.g. Ferland et al. [4]; Laguna et al. [5]). One of the main motivations for this paper is the observation that the design of evolutionary algorithms, including memetic algorithms, makes it particularly difficult to control the balance between intensification and diversification. As Hertz and Widmer [6] point out, preserving the diversity of the population of an EA is necessary. Although EA have the operators to increase or decrease the diversity of the population, most lack the means to *control* this diversification.

This problem is tackled by a new class of EA that we propose: MA|PM or memetic algorithms with population management [7]. *Population management* works by measuring and controlling the diversity of a small population of high-quality solutions, and makes ample use of *distance measures*. The distance measure to use depends on the representation of a solution. For binary problems, the Hamming distance can be used. For permutation problems, several distance measures have been proposed in the literature, e.g. [8–10]. The distance measures used in this paper are based on the *edit distance* (see e.g. Wagner and Fischer [11]), that is well known and easily adaptable to a large number of combinatorial problems [12].

As mentioned, MA|PM (originally called GA|PM or genetic algorithms with population management) are in essence a variant of memetic algorithms [3], but they differ from them in that MA|PM use population management. Other approaches that use a small population of high-quality solutions ([13] call it a pool) include *scatter search* and *path relinking* [14]. MA|PM offer the advantage of being closer to classical EA in terms of algorithm structure and therefore considerably easier to implement. Some work on standard genetic algorithms using a small population is due to Reeves [15]. The use of distance measure for GA design has been proposed for maintaining a diverse population [16] or to locate a set of different solutions of a multimodal problem such as in crowding [17] or fitness sharing [18]. MA|PM differ from these approaches in that they maintain a small population of locally improved solutions and use adaptive population management strategies.

The rest of this paper is structured as follows. Section 2 describes the basic structure of an MA|PM and discusses how distance measures can be used to control the diversity of a population. Some possible population management strategies are also discussed. In Section 3, the algorithm is tested on the multidimensional knapsack problem and the total weighted tardiness single-machine scheduling problems. For both problems, we compare the MA|PM to a hybrid evolutionary algorithm without population management. Although the population management improves the performance of the EA in a big way in both cases, we should remark that the main goal of these applications was to show the effectiveness of population management, not to create solution methods that could compete with the best-performing approaches. That MA|PM can be used to create first-class optimization methods, is being shown in ongoing research. In Sörensen [7], an MA|PM for the vehicle routing problem (VRP) is shown to produce results competitive to the best-known approaches. In Prins et al. [19], an MA|PM is developed for the capacitated arc routing problem (CARP). To date, this MA|PM is the best-known approach for this problem.

2. MA|PM

This section describes the new memetic algorithm with population management. Its main distinguishing features are

- a small population (typically 10–30 solutions),
- a local improvement operator (e.g. a local search, or simple tabu search procedure),
- *population management* to control the diversity of the population.

2.1. Algorithm overview

A memetic algorithm with population management is structured much like a standard memetic algorithm, but differs in the use of population management. An outline is given in Algorithm 1.

Algorithm 1 MA|PM outline

```
1:  initialize population P
2:  set population diversity parameter Δ
3:  repeat
4:    select: p1 and p2 from P
5:    crossover: p1 ⊗ p2 → o1, o2
6:    local search: improve o1 and o2
7:    for each offspring o do
8:      while o does not satisfy conditions for addition (input function)
        do
9:        mutate o
10:     end while
11:     remove solution: P ← P\b
12:     add solution: P ← P ∪ o
13:   end for
14:   update diversity parameter Δ
15: until stopping criterion satisfied
```

Initially, a small population is built randomly or by using initial heuristics. From this population, two *parent* solutions are selected and subjected to the crossover operator, forming one or two new *offspring* solutions. These solutions are improved by a local search operator and added to the population, after being subjected to population management.

2.2. Population management

Population management controls the diversity of a small population of high-quality solutions. It uses an *input function* that determines whether an offspring solution is added to the population or not. In

MA|PM, this input function takes the following two factors into account:

- the quality of the solution, and
- the diversity of the population after addition of the solution, or—in other words—the contribution that the solution makes to the diversity of the population; this is measured as the "distance" of the solution to the population, see Section 2.2.2.

2.2.1. Distance measures

To evaluate whether a candidate solution sufficiently diversifies the population, a distance measure d is used that determines for each pair of solutions their relative distance (or similarity). The distance should be measured *in the solution space* and not—as is commonly done—in the objective function space. Distance measures cannot be developed independent of the problem or even the representation (encoding) of the solution. For binary representations, the Hamming distance can be used. For solutions that are encoded as vectors of real numbers, some form of the Minkowsky-r-distance ($d(\vec{x}, \vec{y}) = (\sum_{i=1}^{n} |x_i - x_j|^r)^{1/r}$) is appropriate (e.g. Euclidean, Manhattan, Chebychev).

For permutation problems, several distance measures have been developed in the literature. An MA|PM developed in this paper (see Section 3.2.1) uses the so-called *edit distance*. The edit distance can be calculated between two strings composed of characters from a finite alphabet. The edit distance is the number of *edit operations* required to transform the first string into the second one. Three *edit operations* are defined: insertion of a character, deletion of a character and substitution of a character by another one. Often, a cost is assigned to each possible edit operation and the edit distance is defined as the minimum total cost of all edit operations required to transform the first string into the second. Using a simple dynamic programming algorithm [11], the edit distance can be calculated in $O(n^2)$. Other, more efficient algorithms have been developed, e.g. Ukkonen [20]. The edit distance can be modified for different types of permutation problems, where solutions cannot be represented as a simple string. An example is the traveling salesman problem, in which a solution does not have a fixed starting position. For a more elaborate discussion, we refer to Sörensen [12].

2.2.2. Distance of a solution to the population

Given a distance measure that can calculate the distance between any pair of solutions, the distance of a given solution s_k to the population can be calculated as follows:

$$\mathrm{d}_P(s_k) = \min_{s_i \in P} \mathrm{d}(s_k, s_i). \tag{1}$$

Calculating the distance of a solution to the population requires calculating $|P|$ distance measures (where $|P|$ is the cardinality of the population). This high computational requirement is one of the reasons why population management techniques are more effective when applied to small populations.

2.2.3. Input function and diversity parameter

It is obvious that a solution that has a small distance to another solution already in the population, will not contribute much to the diversity of a population. Therefore, a solution is not added to the population if its distance to the population is below a certain threshold Δ. We call Δ the *diversity parameter*. Assuming

that the quality of s_k is sufficient, a solution can be added to the population if the following holds:

$$\mathrm{d}_P(s_k) = \min_{s_i \in P} \mathrm{d}(s_k, s_i) \geqslant \Delta. \tag{2}$$

Using the distance $\mathrm{d}_P(s_k)$ and the fitness or objective function value $f(s_k)$, the input function can also use a multi-objective decision method to determine whether s_k should be added to the population or not. A very simple way of doing this (assuming that f should be minimized) is to calculate $f(s_k) + \lambda \mathrm{d}_P(s_k)$. If this value does not exceed a certain threshold, the solution is added, otherwise it is discarded. λ is a parameter that determines the relative importance of the diversity with respect to the solution quality. Of course, more elaborate multi-objective decision methods can be used.

If the local search procedure is effective enough to always ensure the quality of solutions it produces, a solution can be added if Eq. (2) holds, without taking the objective function value of the solution into account.

As shown in Algorithm 1, a solution that does not have a sufficiently large distance to the population, is randomly mutated until it does. Of course, other strategies are possible, such as simply discarding the solution.

2.2.4. Population management strategies

Using the diversity parameter Δ, the diversity of the population can be controlled as higher values for Δ will increase the diversity of the population while lower values will decrease it. A high value of Δ will allow only solutions that have a large distance to all solutions in the population and will lead—perhaps after a few iterations—to a population that consists of very different solutions. A low value of Δ will allow solutions in the population that are relatively similar to solutions already in the population. This will result in a less diverse population.

Several population management strategies can be proposed, using only the diversity parameter Δ. The following lists several potential strategies in increasing order of complexity.

Strategy 1: Δ is set to a constant level. This strategy prevents population convergence, and introduces a constant level of diversification into the population.

Strategy 2: Δ is set to a high level in the beginning of the algorithm. The value of Δ is decreased steadily throughout the working of the algorithm, allowing increased intensification near the end of the run.

Strategy 3: A closely related alternative strategy is to create a few good solutions (low Δ) in the beginning of the algorithm and allow more diversification near the end. A variant of this strategy is used in the experiments.

Strategy 4: Δ is set to a high level in the beginning of the algorithm. It is steadily decreased as better solutions are found. When no improvement of the best solution is found for a set number of iterations, the diversity is increased by increasing Δ, thus introducing new genetic material in the population. After this, Δ is steadily decreased again, etc. This strategy can be called *adaptive* because it uses information about the effectiveness of the search to dynamically control the diversity of the population.

These strategies are graphically represented in Fig. 1. A higher complexity of a population management strategy also implies an increase in the number of parameters that has to be determined. Elaborate strategies like 3 and 4 are especially useful when the search space is large and algorithm runs are long.

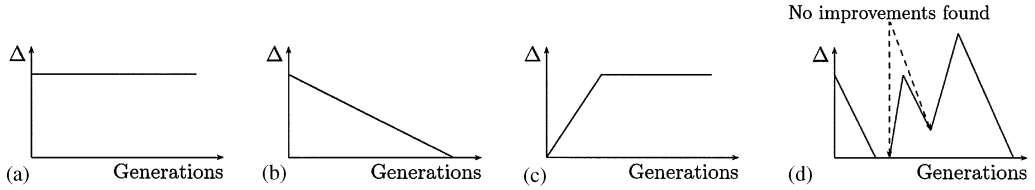

Fig. 1. Population management strategies. (a) Strategy 1; (b) strategy 2; (c) strategy 3 and (d) strategy 4.

2.3. Selection, crossover and other genetic operators

Several genetic operators have to be defined in order for the algorithm to work. The selection operator—used for the selection of parent solutions and for the selection of solutions to remove from the population when new solutions need to be added—can be a binary tournament selection, a roulette wheel selection or any other selection method.

Crossover operators should be designed to preserve the good features of the parent solutions as much as possible. A bad choice of crossover operator will result in offspring solutions that most likely have a rather poor quality.

For the design of efficient selection and crossover operators, we refer to the specialized literature. See e.g. Reeves [21] for a discussion.

2.4. Intensification

A local search procedure is necessary to maintain a population of high-quality solutions. This local search procedure should be able to quickly improve the quality of a solution produced by the crossover operator, without diversifying it into other regions of the search space. Neighborhood-search methods like simple tabu search approaches are particularly useful for this purpose.

3. Experiments

In this section, we apply the principles of MA|PM to two problems: the multidimensional knapsack problem and the total weighted tardiness single-machine scheduling problem. We should note that in neither of the two problems, it is our intention to compete with the best approaches in the literature as this would involve developing specialized crossover operators and advanced local search operators and would distract the attention from the main point: the population management of MA|PM. Instead, we focus on the population management and compare the MA|PM we develop to comparable EA that lack population management.

3.1. The multidimensional knapsack problem

The $\mathcal{NP}$-hard 0/1 multidimensional knapsack problem (MKP01) is a pure 0/1 integer programming problem. Given a set of n items each having an associated *profit* c_i, the objective of this problem is to select the subset of items that maximizes the total profit, while satisfying a set of knapsack constraints.

For each knapsack j, each item i has a given *weight* a_{ij} and the sum of the weights of the chosen items is not allowed to exceed the knapsack capacity b_j. More formally, the MKP01 can be stated as follows:

$$\begin{aligned} &\max \sum_{i=1}^{n} c_i x_i \\ &\text{s.t.} \sum_{i=1}^{m} a_{ij} x_i \leqslant b_j \quad \forall j \in [1, m], \\ &x_i \in 0, 1 \quad \forall i \in [1, n]. \end{aligned} \tag{3}$$

Several approaches have been proposed for this problem, including genetic algorithms [22] and tabu search [23,24].

3.1.1. An MA|PM for the MKP01

A solution of the MKP01 can be represented as a binary string of length n. We set the objective function value of a feasible solution equal to the total profit (i.e. $\sum_i c_i x_i$). For infeasible solutions, we set the objective function value equal to the (negative) total violation of all constraints combined, i.e.

$$\sum_j \left(b_j - \sum_i a_{ij} x_i \right). \tag{4}$$

Since feasible solutions always have a positive evaluation function value and infeasible solutions a negative one, a feasible solution is always preferred and the search will not return an infeasible solution unless no feasible solution could be found. Assigning a negative objective function value to an infeasible solution allows us to use the local search procedure to improve such solutions, assuming that the quality of an infeasible solution is determined by the total violation of the constraints.

A simple *steepest descent* local search procedure is used. This procedure attempts to change the status of every item of a given solution and changes the status of the item that yields the largest increase in objective function value. The search continues until the objective function value cannot be increased by changing the status of a single item.

The distance between two solutions is measured as the Hamming distance between their binary strings. As usual, the distance of a solution to the population is given by the minimum distance of this solution to any solution already in the population.

Selection is done by a binary tournament method: two solutions are randomly chosen and the best one is selected. A simple one-point crossover is used to combine solutions. Each crossover produces two offspring solutions that are locally improved using the local search procedure and then subjected to population management.

Population management determines the distance $d_P(s)$ of a given solution to the population and compares this value to the population diversity parameter Δ. If this distance is smaller than Δ, the solution s is mutated. This is done by randomly flipping $\Delta - d_P(s)$ bits of s. This procedure does not ensure that the mutated solution has a sufficiently large distance to the population, but it diversifies it without causing too much overhead.

The performance of the MA|PM is compared to the performance of the evolutionary algorithm that results from removing the population management component from the MA|PM and replacing it with random mutation. From now on, we refer to this procedure as HGA (hybrid GA).

Table 1
Comparison of MA|PM and HGA for the multidimensional knapsack problem

	Nr.opt.	Min. 1 opt.	Avg. it.	Avg. CPU
MA|PM strategy 1	3.84	87.27%	3537	6.43
MA|PM strategy 3	3.73	92.59%	3879	10.00
HGA	0.51	25.45%	9206	9.38

3.1.2. Experiments

The MA|PM is tested on 48 well-known test problems available from the OR library (http://www.ms.ic.ac.uk/jeb/pub/mknap2.txt).

A first experiment uses the simplest population management for the MA|PM, i.e. $\Delta = 1$. A second experiment uses more advanced population management (a variant of strategy 3 in Fig. 1) with

$$\Delta = \frac{\text{current iteration}}{\text{maximum iterations}} \times \frac{n}{5},$$

where n is the number of items. This strategy sets Δ to 0 in the beginning of the algorithm and linearly increases it to a maximum of $n/5$.

A population of 10 is used in all experiments. The HGA uses a 10% random mutation rate. Each algorithm is awarded 10,000 generations to find the optimal solution. Time is measured when the optimal solution is found or at the end of the 10,000 generations when it is not found. Each experiment is repeated five times.

3.1.3. Numerical comparison

Results show that the MA|PM outperforms the HGA. The average quality over the five runs of the solutions produced by the MA|PM is better in all cases. In 46 out of 55 cases (84%), the worst solution found by the MA|PM is better than the best solution found by the HGA. More detailed results can be found in Table 1.

This table should be read as follows. "Nr. opt." is the average number of times the optimal solution was found over the five runs, "Min. 1 opt." is the average percentage of instances for which the optimal solution was found at least in one of the 5 runs, "Avg. it." is the average number of iterations, and "Avg. CPU (s)" is the average time required in seconds.

As can be seen, the MA|PM performs much better than the HGA without population management. The average time per iteration is somewhat higher, but the solutions found are much better and the MA|PM therefore performs less iterations. The more advanced population management increases the average number of iterations required and also the average CPU time. It does however improve the robustness of the algorithm in that the optimal solution is now found in over 92% of the problem instances.

3.2. The total weighted tardiness single-machine scheduling problem

A set of n jobs has to be sequenced on a single machine. Preemption is not allowed. Jobs are not available before a release date r_j and are processed for p_j units of time. For each job, a due date d_j and a weight w_j is given. If C_j denotes the completion time of job j, the tardiness is defined by $T_j = \max(0, C_j - d_j)$.

The objective is to minimize the total weighted tardiness ($1|r_j|\sum w_jT_j$). This problem is $\mathcal{NP}$-Hard in a strong sense.

The total weighted tardiness problem with release dates is probably one of the most difficult one-machine scheduling problem. If all the weights are equal to one ($1|r_j|\sum T_j$) the problem is $\mathcal{NP}$-Hard in a strong sense [25]. For different weights and the release dates all equal to zero ($1||\sum w_jT_j$), problem is also $\mathcal{NP}$-Hard in a strong sense [26].

Recent approaches on the $1||\sum w_jT_j$ problem are proposed in the literature. In Crauwels et al. [27], a comparison between simulated annealing, tabu search and genetic algorithms is discussed and in Congram et al. [28] the iterated dynasearch algorithm is presented and shown to yield the best solutions. Results are reported in the OR-Library. A recent constraint-based approach [29] can solve problem instances with up to 40 jobs.

3.2.1. A MA|PM for the total weighted tardiness problem

In this section, we develop a simple MA|PM for the total weighted tardiness problem and compare it to a similar hybrid GA that lacks population management. To perform a fair comparison between the two approaches, common components are used for the HGA and MA|PM algorithms.

The *selection* process used is a binary tournament selection, the *crossover* operator is the standard LOX operator [30] and the *mutation* operator is the GPI (general pairwise interchange). The local search procedure is also based on the GPI and a steepest descent algorithm is applied. The algorithm is stopped after 60 s of CPU time.

For the numerical experiments, we use a variation of strategy 3. The distance measure used is the *edit distance* discussed in Section 2.2.1. In the implementation, Δ is measured in percentage (result of the distance measure divided by the maximum distance value). At the initialization step, Δ is fixed to 1%. Every 15 iterations, Δ is multiplied by a growth factor (1.05). When an upper limit of 50% is reached, Δ is not increased anymore, since this would not allow the algorithm to find a new solution that is accepted during the search.

In the HGA, not all solutions are subjected to local search as this almost always induces premature convergence. Instead, the local search rate p_{ls} is set to 10%, the mutation rate p_{m} is fixed to 10%. The size of the population is set to 25. For the MA|PM, the size of the population is only 10 individuals.

3.2.2. Numerical comparison

Results are reported in different tables. Table 2 summarizes the results for the ODD instances (these instances can be obtained by email request). Optimal solutions are not known for this set. The first column indicates the set of instances (ODD20 is the ODD instances with 20 jobs), the following three columns provides the results for the GA|PM method and the next three columns, the results for the HGA method. Note that for a fair comparison, the same amount of time (60 s) has been given to the MA|PM and the HGA.

For each of the methods, the first column gives the number of times the method reports the best solution (among the two methods), the second column is the average deviation from the best solution. The last column gives the average number of iterations done in 60 s.

The OR-Library instances are generated for the $1||\sum w_jT_j$ problem and can be solved by our approach. For 40-job instances, 124 out of 125 optimal solutions are provided on the OR-Library web site. For 50-job instances, 115 out of 125 solutions are known. When the optimal solution is unknown, best solutions provided by Crauwels et al. [27] are used instead.

Table 2
Numerical results for ODD instances, CPU time = 60 s

Set of inst.	MA|PM results			HGA results		
	First pos.	Avg. gap (in %)	Avg. iter.	First pos.	Avg. gap (in %)	Avg. iter.
ODD20	0	0.000	32375	0	0.000	952757
ODD40	0	0.000	4142	0	0.000	258526
ODD60	6	0.088	1685	1	0.251	88106
ODD80	6	0.003	1153	1	0.083	39040
ODD100	8	0.064	844	4	0.118	19854
Global	20	0.057	8040	6	0.148	271657

Table 3
Numerical results for OR-Library instances, CPU time = 60 s

Method used	Opt. sol.	First pos.	Avg. gap (in %)	Iterations	
				Avg.	Max
ORLib40 results					
MA|PM	125	4	0.000	5682	8961
HGA	121	0	0.284	327763	470984
ORLib50 results					
MA|PM	123	11	0.002	3334	5298
HGA	113	0	0.595	196024	287802
ORLib100 results					
MA|PM	94	27	0.276	1030	1480
HGA	85	16	2.110	34164	56521

In Table 3 the column "Opt. sol." ("Best sol." for 100 job instances) counts the number of times the algorithm find the optimal solution (or best known). The rest of the columns is identical to the previous table except that the deviation is measured from the optimal solution. The maximum number of iterations is added too. Again, the same amount of time has been allocated to the two methods for a fair comparison.

These results show that the MA|PM approach performs better than the HGA approach, even though it operates on a smaller population. For the OR-Library instances, more optimal solutions are found and the approach is ranked first in more cases. Moreover, the deviation from the optimal (or best known) solution is always smaller with the proposed method. When we use the same stopping conditions (fixed number of iterations without improvement of the best solution) the difference between the two algorithms is even greater, but the MA|PM requires more time.

4. Conclusions

It is commonly known that the performance of an EA (and any metaheuristic) on a hard combinatorial optimization problem depends in a big way on the balance between intensification and diversification. Although all EAs possess the necessary operators for intensification and diversification, many EA implementations lack a mechanism to control the balance between these two factors. This paper introduced a new class of evolutionary algorithms, MA|PM or memetic algorithms with population management, that addresses this issue. MA|PM are hybrid EAs that—like scatter search—operate on a small population of high-quality individuals. The main distinguishing feature of an MA|PM is the application of *population management strategies* to control the diversity of the population. We discussed how such strategies can be easily implemented by controlling the value of only a single parameter, the population diversity parameter Δ.

The proposed approach was tested by applying it to two different combinatorial problems: the multidimensional knapsack problem (a pure 0/1 integer programming problem) and the total weighted tardiness single-machine scheduling problem, a difficult permutation problem. We discussed that different problems require different distance measures and proposed to use the edit distance for permutation problems. For binary problems, the Hamming distance was used. Numerical comparison showed that the MA|PM outperformed very similar hybrid genetic algorithms without population management (but with random mutation instead).

References

[1] Holland JH. Adaptation in natural and artificial systems. Technical report, University of Michigan, Ann Arbor, 1975.
[2] Goldberg DE. Genetic algorithms in search, optimization and machine learning. Reading, MA: Addison-Wesley; 1989.
[3] Moscato P. On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Technical report C3P 826, Caltech Concurrent Computation Program, 1989.
[4] Ferland JA, Ichoua S, Lavoie A, Gagné E. Scheduling using tabu search methods with intensification and diversification. Computers and Operations Research 2001;28:1075–92.
[5] Laguna M, Martí R, Campos V. Intensification and diversification with elite tabu search solutions for the linear ordering problem. Computers and Operations Research 1999;26:1217–30.
[6] Hertz A, Widmer M. Guidelines for the use of meta-heuristics in combinatorial optimization. European Journal of Operational Research 2003;151(2):247–52.
[7] Sörensen K. A framework for robust and flexible optimization using meta-heuristics with applications in supply chain design. PhD thesis, University of Antwerp, 2003b.
[8] Ronald S. Distance functions for order-based encodings. In: Fogel D, editor. Proceedings of the IEEE conference on evolutionary computation. New York: IEEE Press; 1997. p. 641–6.
[9] Ronald S. More distance functions for order-based encodings. In: Proceedings of the IEEE conference on evolutionary computation. New York: IEEE Press; 1998. p. 558–63.
[10] Campos V, Laguna M, Martí, R. Context-independent scatter and tabu search for permutation problems. INFORMS Journal on Computing, 2003, to appear. URL http://leeds.colorado.edu/Faculty/Laguna/publications.htm
[11] Wagner RA, Fischer MJ. The string-to-string correction problem. Journal of the Association for Computing Machinery 1974;21:168–73.
[12] Sörensen K. Distance measures based on the edit distance for permutation-type representations. In: Barry A, editor. Proceedings of the workshop on analysis and design of representations and operators (ADoRo), GECCO conference. Chicago, 2003a. p. 15–21.
[13] Greistorfer P, Voß S. Controlled pool maintenance in combinatorial optimization. In: Rego C, Alidaee B, editors. Adaptive memory and evolution: tabu search and scatter search. Boston: Kluwer Academic Publishers; to appear.

[14] Glover F. A template for scatter search and path relinking. In: Hao J-K, Lutton E, Ronald E, Schoenauer M, Snyers D, editors. Artificial evolution, volume 1363 of lecture notes in computer science. Berlin: Springer; 1998. p. 13–54.
[15] Reeves CR. Using genetic algorithms with small populations. In: Forrest S, editor. Proceedings of the fifth international conference on genetic algorithms. San Mateo: Morgan Kaufmann; 1993. p. 92–99.
[16] Mauldin M. Maintaining diversity in genetic search. In: Proceedings of the national conference on artificial intelligence. 1984. p. 247–50.
[17] Mahfoud SW. Crowding and preselection revisited. In: Manner R, Manderick B, editors. Parallel problem solving from nature. Amsterdam: Elsevier; 1992. p. 27–36.
[18] Goldberg D. Genetic algorithms with sharing for multimodal function optimization. In: Proceedings of the second international conference on genetic algorithms. 1987. p. 41–9.
[19] Prins C, Sevaux M, Srensen K. A genetic algorithm with population management for the CARP. In: Proceedings of TRISTAN V, Guadeloupe, 2004.
[20] Ukkonen E. Finding approximate patterns in strings. Journal of Algorithms 1985;6:132–7.
[21] Reeves CR. Genetic algorithms for the operations researcher. INFORMS Journal on Computing 1997;9:231–50.
[22] Chu PC, Beasley JE. A genetic algorithm for the multidimensional knapsack problem. Journal of Heuristic 1998;4:63–86.
[23] Dammeyer F, Voß S. Dynamic tabu list management using the reverse elimination method. Annals of Operations Research 1993;41:31–46.
[24] Glover F, Kochenberger GA. Critical event tabu search for the multidimensional knapsack problem. In: Osman IH, Kelly JP, editors. Metaheuristics: the theory and applications. Boston: Kluwer Academic Publishers; 1996. p. 407–27.
[25] Garey MR, Johnson DS. Computers and intractability: a guide to theory of np-completeness. San Franscisco, USA: Freeman; 1979.
[26] Lawler EL. A 'pseudo-polynomial' algorithm for sequencing jobs to minimize total tardiness. Annals of Discrete Mathematics 1977;1:331–42.
[27] Crauwels HAJ, Potts CN, Van Wassenhove LN. Local search heuristics for the single machine total weighted tardiness scheduling problem. Informs Journal of Computing 1998;10(3):341–50.
[28] Congram R, Potts CN, van de Velde S. An iterated Dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. Informs Journal on Computing 2002;14(1):52–67.
[29] Baptiste P, Jouglet A, Carlier J. A branch-and-bound procedure to minimize total tardiness on one machine with arbitrary release dates. European Journal of Operational Research, 2004, to appear.
[30] Portmann M.C. Genetic algorithm and scheduling: a state of the art and some proposition. Proceedings of the workshop on production planning and control. Mons, Belgium, September 9–12, 1996.