Genetic and memetic algorithms for scheduling railway maintenance activities

Gabriella Budai*, Rommert Dekker† and Uzay Kaymak†

December 16, 2009

Econometric Institute Report EI 2009-30

Abstract

Nowadays railway companies are confronted with high infrastructure maintenance costs. Therefore good strategies are needed to carry out these maintenance activities in a most cost effective way. In this paper we solve the preventive maintenance scheduling problem (PMSP) using genetic algorithms, memetic algorithms and a two-phase heuristic based on opportunities. The aim of the PMSP is to schedule the (short) routine activities and (long) unique projects for one link in the rail network for a certain planning period such that the overall cost is minimized. To reduce costs and inconvenience for the travellers and operators, these maintenance works are clustered as much as possible in the same time period. The performance of the algorithms presented in this paper are compared with the performance of the methods from an earlier work, Budai *et al.* (2006), using some randomly generated instances.

Keywords: heuristics, maintenance optimization, opportunities, genetic algorithm, memetic algorithm

1 Introduction

Nowadays railway companies are confronted with high infrastructure maintenance costs. For example, in the Australian freight operations the maintenance costs represent 25-35% of total train operating costs (Higgins (1998)) and in the Netherlands in 2003 the expenses of maintenance and renewal were approximately 295 million Euro for maintenance and 150 million Euro for renewal (Swier (2003)). Therefore good strategies are needed to carry out these maintenance activities in a most cost effective way. Besides the high maintenance costs, the other problem that the railway infrastructure manager is facing is that due to densely used railway network it is more and more difficult to find time periods for preventive infrastructure maintenance that are long enough. Moreover, these time periods should be chosen such that the train operation is not disturbed too much.

The IMPROVERAIL project(see Improverail (2002)) and Budai and Dekker (2002) show that preventive railway maintenance works are carried out in several countries during train service. In the actual train timetable possible possession allocations are scheduled for maintenance so that it should not affect regular train services too much. This can be done, however, for occasionally used tracks, which is the case in Australia and some European countries. Some important references in this respect are (Higgins (1998) and Cheung *et al.* (1999)). If tracks are used frequently, one has to perform maintenance during nights, when the train traffic is almost absent, or during weekends (with possible interruption of the train services), when there are fewer disturbances for the passengers (see *e.g.* Budai *et al.* (2006)). In that case one can either make a cyclic static schedule, which is made by Den Hertog *et al.* (2005) and Van Zante-de Fokkert *et al.* (2007) for the Dutch situation or a dynamic schedule with a rolling horizon, which has to be made regularly. The latter is described in Cheung *et al.* (1999).

The Preventive Maintenance Scheduling Problem in the literature

Initially, Budai *et al.* (2006) started to study the preventive maintenance scheduling problem (PMSP) for railway infrastructure, where a schedule for preventive maintenance activities has to be found for one link

*Ortec bv, Groningenweg 6k, 2803 PV Gouda, The Netherlands. E-mail: Gabriella.Balke@ortec.com

†Econometric Institute, Erasmus University Rotterdam, P.O. Box 1738, NL-3000 DR, Rotterdam, The Netherlands. E-mail: {rdekker,kaymak}@few.eur.nl

such that the sum of the track possession costs and maintenance costs is minimized. The possession costs are mainly determined by the possession time, which is the time that a track is required for maintenance and cannot be used for railway traffic. The preventive maintenance activities consist of small *routine works* and *projects*. The routine works (*e.g.* inspections or small repairs) are short, but frequent. They are scheduled from once per month to once in a year. The projects include longer renewal works (*e.g.* sleeper tamping, ballast cleaning), that are in general performed only once or twice every few years and they are triggered by condition measurements. Although the PMSP considers one rail link only, it can be extended to a network using the concept of the Single Track Grids (STGs) presented in Van Zante-de Fokkert *et al.* (2007).

Budai *et al.* (2006) show that the PMSP is an NP hard problem. Moreover, the authors provide a mathematical programming formulation for PMSP and using this formulation and the CPlex solver they attempt to find the optimal solution for their problem. In general the computation time to find the optimal solution is too high, therefore some greedy heuristics were developed. These heuristics give a feasible solution within a very short time. However, even the best heuristic sometimes differs up to 7% from the value of the best solution found using the exact method. Therefore, our objective is now to generate better results for the PMSP using other type of solution methods.

Several authors recommend using genetic (GA) and memetic (MA) algorithms for scheduling problems. Both algorithms are popular in maintenance applications because of their robust and fast search capabilities that help to reduce the computational complexity of large optimisation problems, such as large scale maintenance scheduling models. A detailed description of GA and MA is presented in Section 3.2 and Section 3.3. Here we explain briefly only some successful applications of GA and MA for maintenance scheduling problems, alike the PMSP.

Genetic algorithms in the maintenance scheduling literature

GAs, that is based on the principles of natural selection and genetics, were developed by Holland (1975) and are widely used since then by many researchers in different fields. Here we list some studies on maintenance optimization where GA was used with success.

Grimes (1995) deals with the problem of planning a specific track maintenance work, the track tamping, using a genetic algorithm. GA gives good results for short sections of track (10 miles) and poor results for long sections (50 miles). In Sriskandarajah *et al.* (1998) GA is used for scheduling the frequency-based overhaul maintenance of the rolling stock (*i.e.* trains). The computational results show that the GA gives close to optimal solutions for randomly generated problems with known optimal solutions. GA is used in Fwa *et al.* (1994), and later in Chan *et al.* (2001), for planning road-maintenance. Liu *et al.* (1997) and Morcous and Lounis (2005) present two different approaches for optimizing maintenance strategies of bridge deck networks. Moreover, GA proves to be a very powerful tool for maintenance scheduling in power systems too. Some important works in this respect are Negnevitsky and Kelareva (1999), Abdulwhab *et al.* (2004). In Munoz *et al.* (1997), Lapa *et al.* (2000) and Lapa *et al.* (2006) GA is used for maintenance scheduling in the nuclear power plan.

Memetic algorithms in the maintenance scheduling literature

In many articles in the literature the performance and effectiveness of GAs is often improved by incorporating a local search operator (e.g. tabu search, simulated annealing, hill climbing) into the GA by applying the operator to each member of the population after each generation. These approaches are called in the literature memetic algorithms (MAs). Here we highlight a couple of promising applications of the MAs in maintenance optimization.

Dahal and Chakpitak (2007) presents the application of the genetic algorithms, simulated annealing (SA) and their hybrid for generator maintenance scheduling in power systems. They show that the hybrid approach is less sensitive to variation of the GA and SA parameters and gives better averaged results than GA and SA. Burke *et al.* (1997a) deals with the thermal generator maintenance scheduling problem, where the maintenance of a number of thermal generator units is scheduled such that the maintenance cost is minimized and enough capacity is provided to meet the anticipated demand. This problem was earlier solved by traditional optimization techniques, such as integer programming, dynamic programming and branch and bound. For small problems these methods gave an optimal solution, but as the size of the problem increased, the size of the solution space increased exponentially and hence also the running time of these algorithms. To overcome this problem, in Burke *et al.* (1997a), simulated annealing, genetic algorithms, tabu search and a combination of simulated annealing and tabu search were implemented to solve the thermal generator maintenance scheduling problem. Their results show that the tabu search al-

gorithm performs the best. The genetic algorithm was the worst performing algorithm for solutions with large numbers of feasible solutions, but performs slightly better than simulated annealing for problems with a small number of feasible solutions. Burke and Smith (1997b) solves the same problem by using memetic algorithm, i.e a genetic algorithm combined with tabu search. For small problems the memetic algorithm performs as good as simulated annealing and tabu search, but for large problems memetic algorithm outperforms both algorithms. In Burke and Smith (2000) the thermal generator maintenance scheduling problem is solved by using the memetic algorithm with three types of local search, namely simulated annealing, hill climbing and tabu search. The memetic algorithm with tabu search proves again to be the best method, followed by the memetic algorithm with hill climbing. The memetic algorithm with simulated annealing gives reasonable results, but the execution time is significantly higher than the execution times of the other two algorithms. The computational results are also promising when a memetic algorithm with tabu search, simulated annealing and hill climbing is used for solving maintenance scheduling problems for the National Grid in South Wales (see *e.g.* Burke and Smith (1999b)). Li *et al.* (2002) use a combination of genetic algorithm with tabu search for solving maintenance scheduling problem of oil storage tanks. It has been shown that this tabu-based genetic algorithm outperforms GA.

Due to the above described successful applications of the meta-heuristics on problems alike the PMSP, in this paper we will focus on finding better results for PMSP using the GA and MA with three local search algorithms, namely the *Steepest Hill Climbing (SHC)*, *Simulated Annealing (SA)* and *Tabu Search (TS)*. Moreover, we develop a new method, called GA_OPP, where the preventive maintenance works are carried out at opportunities that are generated either randomly or by an already planned preventive maintenance work. GA_OPP is a two phase method. In the first phase opportunities are generated using GA and in the second phase all the executions of the preventive works are fitted as much as possible to these opportunities.

The idea of performing preventive works at opportunities has already been used in Budai *et al.* (2006) at the *Opportunity Based Heuristic (OBH)*. In that paper the execution times of the most frequent routine work were used as opportunities for performing the other routine works as well. Savic *et al.* (1995a) and Savic *et al.* (1995b) formulate the opportunity based maintenance problem (OBM) as a set partitioning problem and solved it using GAs. The OBM is somewhat different than OBH, since in OBM the components in a system are divided into groups and as soon as a component of a group fails, all the components of the group are replaced. Hence, a failure of a component of a group is used as an opportunity to replace the rest of the components of this group. Thus, the problem is to find an optimal grouping of the components of a system such that the total maintenance cost is minimized. In Dekker and van Rijn (1996) a decision-support system (PROMPT) for opportunity-based preventive maintenance is discussed. PROMPT was developed to take care of the random occurrence of opportunities of restricted duration. Here, opportunities are not only failures of other components, but also preventive maintenance on (essential) components.

It is worth to mention that the PMSP is just a basic problem, but it can be easily extended to other types of practical problems, for instance scheduling the maintenance works while taking the available manpower into account.

The remainder of the paper is organized as follows. In the next section we give a short problem description, followed by Section 3 where we discuss the solution approaches for PMSP, such as Opportunity Based heuristic, Most Costly Work First heuristic, Genetic algorithms, Memetic algorithms and the Two-phase opportunities based heuristic. The computational results are presented and analyzed in Section 4. Finally, in Section 5 and we formulate our conclusions.

2 Description of the Preventive Maintenance Scheduling Problem (PMSP)

The aim of the PMSP is to give a schedule for preventive maintenance activities in a finite horizon, such that the maintenance works are clustered as much as possible in the same period and the overall cost is minimized. Clustering multiple maintenance activities in the same period leads to a reduction in the possession costs, since execution of a group of works requires only one track possession. Moreover, since maintenance works often require one or more set-up activities, such as crew and equipment travelling, carrying out multiple works simultaneously results in significant savings in the set-up costs.

The PMSP can be defined as follows. Consider a set of routine maintenance activities and projects over a finite planning horizon which is divided into a number of periods (*e.g.* weeks, months). For each routine work the *planning cycle* and the *age* is known. The planning cycle is equal to the maximum

number of time periods between two consecutive executions. The age is defined as the number of time periods elapsed since the routine work was carried out for the last time. The duration, the earliest and latest possible starting times of each project are known as well. Moreover, some routine works and projects may be combined to reduce the track possession times, but others may exclude each other. We assume that a list of works (routine works and/or projects) that can be combined is given. The *maintenance costs* of each routine work and project and the costs of having a *track possession* in the planning period are known. Since our planning horizon is finite and the routine activities are repetitive, an end of horizon valuation is used. Thus, we assume that a *penalty* is paid if the last execution of the routine works is carried out too early compared to the end of planning horizon. Thus, the penalty cost for a given routine work is equal to its maintenance cost times the length of the remaining interval from the last execution until the end of planning horizon divided by the length of the planning cycle of this routine work. The goal of the PMSP is to schedule the given set of routine maintenance works and projects, such that the sum of track possession costs, maintenance costs and the penalties paid for too early executions is minimized.

The main problem with the PMSP is that we have a combination of repetitive work and once-only work. If we would have only repetitive work, then one could look for structure in the problem. An example could be power-of-two policies, where a base interval has been selected and every activity is executed at a power of two of this base interval. Roundy (1985) has shown that in case of inventory control the maximum loss of such a policy compared to the optimal policy is only 6%. However, the loss functions in case of inventory control are much flatter than in our case, so such a policy is not likely to perform well, especially if regular work also has to be combined with projects.

3 Solution approaches for PMSP

In this section we attempt to develop new techniques that generate better results for the PMSP than the methods used in Budai *et al.* (2006). First we recall briefly the already developed methods and after that we focus on GAs, MAs and the Two-phase opportunities based heuristic.

3.1 Solution approaches in Budai *et al.* (2006)

Budai *et al.* (2006) present an exact method and two heuristics designed for the PMSP, namely the Opportunity Based heuristic (OBH) and Most Costly Work First heuristic (MCWF). Both heuristics are greedy in the sense that they try to combine every maintenance work together.

In the OBH the execution times of the most frequent work create opportunities to carry out the other routine works as well. The most frequent work is carried out at exactly equivalent intervals and for all the other routine works it is checked whether it is cost effective to use these already created opportunities or rather creating own opportunities requiring more possessions. Finally, the projects are scheduled to the best place given the schedule of the routine works.

In the MCWF the execution times of the most costly work and half of the time intervals between two subsequent executions are used as opportunities for execution times of the other routine works. The restriction here is that each work can be carried out only at these opportunities, no matter whether it leads to a cost reduction or not. The projects are finally scheduled to the best place given the schedule of the routine works. A detailed description of these approaches can be found in the referenced article.

3.2 Genetic Algorithm (GA)

First we give a general description of the GAs, followed by a section where we describe the encoding of the chromosomes, the creation of the initial population, the fitness evaluation and the selection schemes used for the PMSP. Finally, we give a description of the implemented crossover and mutation operators.

3.2.1 General description of GA

Genetic algorithms use a direct analogy of natural behaviour. A solution of the problem is represented as a chromosome, using strings of binary digits. The process of translating a certain problem into chromosomes is called encoding. Some chromosomes are better than others, thus to test the performance of an individual chromosome, a fitness function is needed. The total number of chromosomes is called the population size. After an initial population is created, the crossover and mutation operators are used to get the next generations.

Crossover is the analogue of mating in nature. From the population two parents are chosen with a fitness dependent chance. Crossover of these two parents result into two children (offsprings). Because parents with good chromosomes are more likely to be selected for crossover, the average fitness of the children will generally be higher than that of the population of parents. Thus selecting the fittest parents, the good characteristics will survive over generations. This means that a number of iterations the population will converge to an optimal solution to the problem. To avoid that the process gets stuck in a local optimum, mutation of the chromosomes is used. This will guarantee diversity among the solutions and so genetic algorithms find global optima. To control the genetic process, several parameters need to be chosen. These are: the population size, number of iterations, the probabilities for the genetic operators, *etc.*

3.2.2 GA for PMSP

In the genetic algorithm used for the PMSP the chromosomes are represented as two dimensional binary arrays, where columns represent the weeks and rows represent the different routine works and projects. Thus an 1 at position (i, j) in the chromosome means that routine work i is planned for time period j.

In Figure 1 (figure originates from Negnevitsky (2005)), we present the steps that we follow in the GA used for the PMSP. These are actually the common steps used in the literature of GA (see *e.g.* Negnevitsky and Kelareva (1999), Sastry *et al.* (2005)). Each step is explained in details below.

Step 1: Create initial population

In *Step 1*, the initial population for the PMSP is created randomly in the following way. For routine works, the executions are determined by choosing at random a number between three-quarters of the planning cycle and the entire planning cycle. For the first execution from this selected number, the age will be subtracted in order to not violate the planning cycle restriction. If this number is negative, the first week is chosen. For projects, since the duration is fixed, choosing a starting period is sufficient. This period is chosen at random in the set of possible starting points, but taking into account that each project has to be finished before the end of the planning period.

Step 2: Fitness evaluation

In order to assign in *Step 2* a fitness to every chromosome, the total costs for the resulting solutions are computed. The total cost is the sum of the maintenance costs, possession costs and penalties resulting from the end-of-horizon validation. As we already mentioned in Section 2, some maintenance works cannot be performed together at the same time period. Unfortunately, it is not possible to be add this as a restriction to the GA. Therefore, in order to prevent solutions that violate such a combination we add to the overall costs a very high penalty cost each time that not combinable works are planned for the same time period. Since PMSP is a minimization problem and the GA maximizes fitness, the fitness is calculated as a reciprocal of the total cost.

Step 3: Selection scheme

If the maximum number of iterations is reached, the algorithm stops, otherwise we proceed with *Step 3*, namely with selecting two chromosomes for mating. The selection of the chromosomes for producing offsprings is done by a selection scheme, such as the rank based roulette wheel selection, the tournament selection, *etc.* With the rank based roulette wheel selection, the chromosomes are ranked according to their fitness value, where the chromosome with the lowest fitness gets rank 1 and the one with the highest fitness gets a rank equal to the population size. The probability that a chromosome is selected is based on his rank. With the tournament selection, two chromosomes are selected at random. The one that has the highest fitness is selected as parent. After that two new chromosomes are selected at random in order to find the other parent.

Step 4: Crossover

One way to create offsprings from the chosen two chromosomes is by *one point crossover with indivisible parts*. Indivisible parts implies that the individual schedules of every routine work or project are kept equal throughout crossover, so that only the combination of the different routine works and projects is changed. Figure 2 gives an example of one-point crossover with indivisible parts.

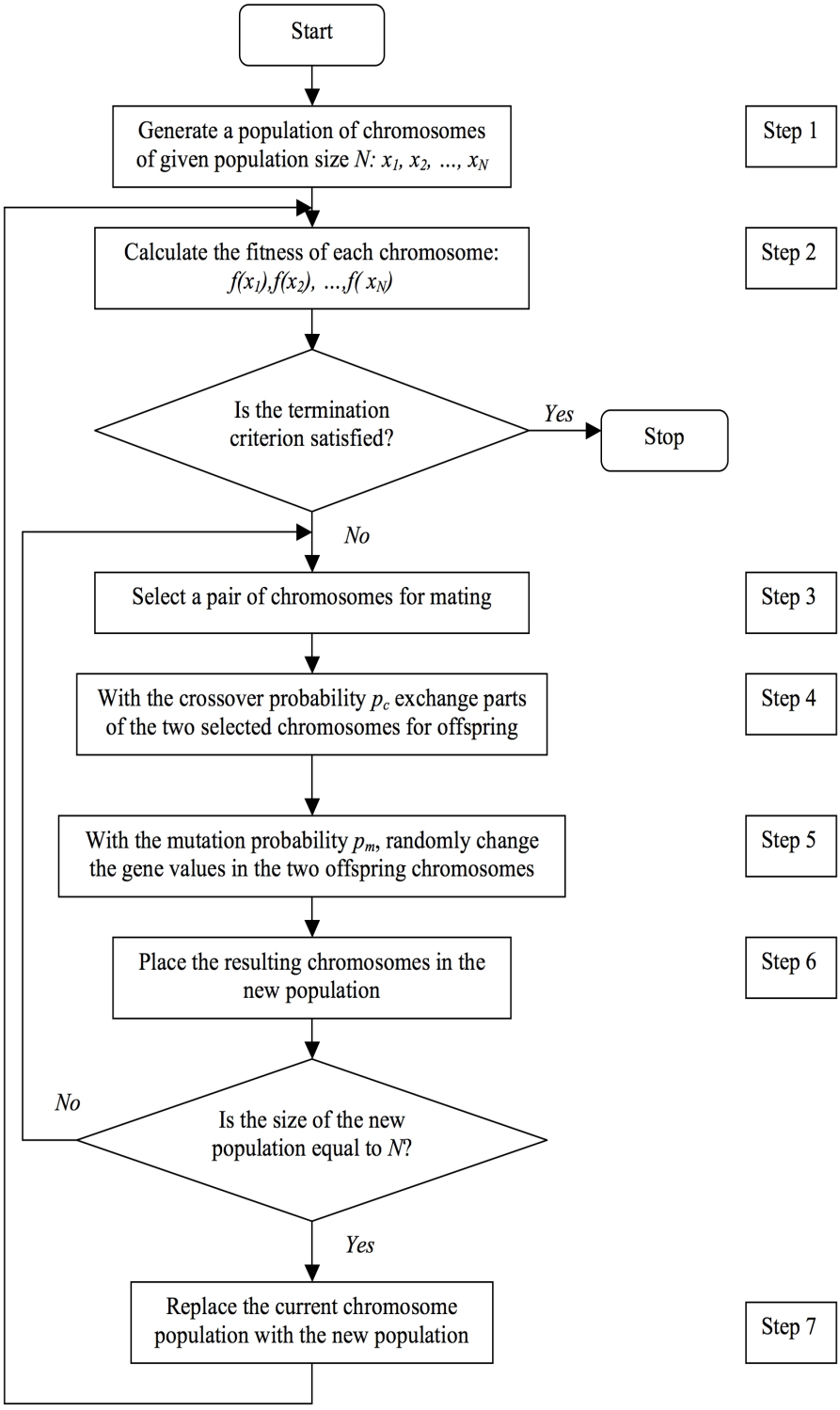

Figure 1: Schematic overview of a genetic algorithm (Negnevitsky (2005))

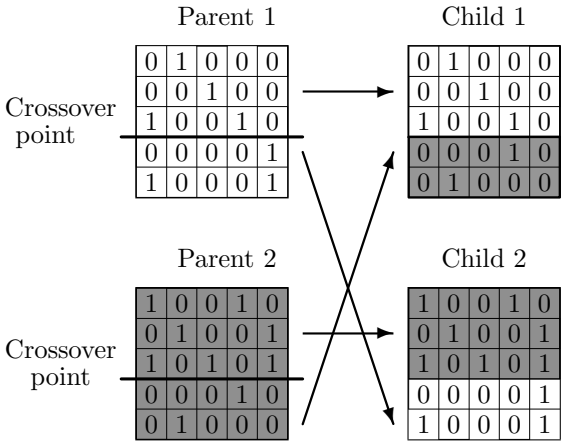

Figure 2: Crossover for GA

Step 5: Mutation

The other way to create offsprings is by using mutation operators. We recall, that the purpose of the mutation in the GA is to allow the algorithm to avoid local minima by preventing the population of chromosomes from becoming too similar to each other, thus slowing or even stopping evolution. We use five types of mutation operators. Three operators can be used for the routine works, one for projects and the fifth to the entire chromosome. These five mutation operators are described below.

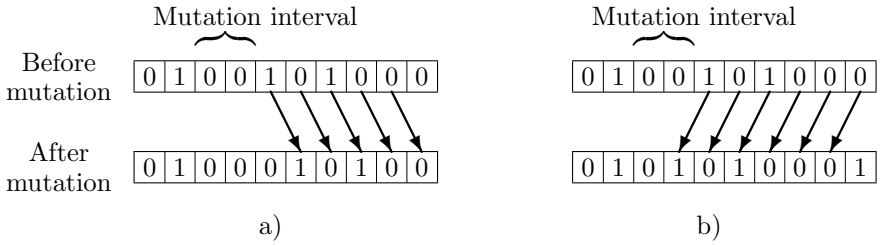

Figure 3: Zero insertion and zero removal mutation

The *zero insertion operator* (Figure 3a) tries to insert one or more zeros in one of the intervals between two executions of a routine work. First, it checks whether it is possible to insert a zero in any of the intervals, thus whether the second execution can be delayed one or more time periods. If this is not possible, there will be no changes. Otherwise, one of the intervals is selected at random and in this interval the operator checks how many zero's can be inserted in that interval at most. The number of zeros that will be inserted is a random number between one and the maximum number of zeros that can be inserted. After insertion all the elements after the inserted zero(s) are shifted to the right. If it is not possible to insert a zero in the random interval, a new random interval is chosen.

The *zero removal mutation operator* (Figure 3b) chooses an interval at random and in this interval a random number of zero's is removed. Thus, a number of executions of a routine work are carried out earlier than it was originally planned. If the last interval becomes too large due to the removal of some zero's, a new random execution is added.

The *single one shift mutation operator* (Figure 4a) chooses a random execution of a routine work and it checks whether it is possible to execute this routine work earlier or later. If this is possible then a random direction is chosen (forward or backward) and the execution is moved either forward or backward with a random number of places. All the other executions remain unchanged.

The three mutation operators described above only work on the routine maintenance works. They are actually shift operators, since the executions are shifted random time periods earlier or later. *Projects* have their own type of *mutation*, which chooses randomly a new starting time in the given (earliest allowed staring time, latest allowed staring time) interval. This is illustrated on Figure 4b.

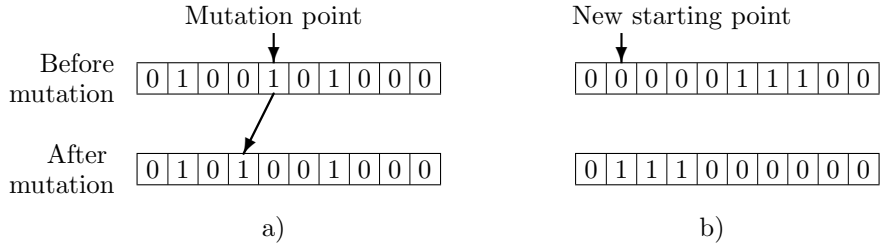

Figure 4: Single one shift and project mutation

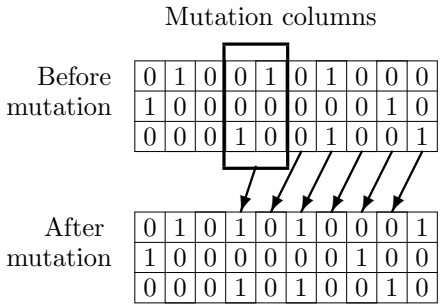

Figure 5: Join together columns mutation

The *join together columns mutation* (Figure 5) can be used for the whole chromosome, affecting all the routine works and projects. The following steps are used by this mutation operator:

a) Check whether there are two consecutive weeks with at least one execution of different works. If no, exit the algorithm. If yes, then one of these weeks is selected randomly and the corresponding columns are joined together in the chromosome.

b) Move all the executions after the merged columns one time period forward.

c) If the last interval for the routine works exceeds the cycle length, then the last column is filled with one, otherwise with zero.

d) Check if the starting times of all the projects are still in their allowed intervals. If not, reschedule the projects as it is described at the project mutation.

The crossover and mutation operators are performed with user defined probabilities. After selecting two parents out of the current population, it is first determined whether crossover will be performed on these parents (Step 4). If so, these two parents are replaced by their two children. After that, either both children or both parents (if the crossover was not selected) will be mutated individually (Step 5). The join columns together mutation is a global operator, so it cannot be used together with the other mutation operators, while the other four operators can be performed simultaneously on the same chromosome. Therefore, based on the user defined probability it is first determined whether the join columns together mutation will be performed on the chromosome. If not, it is decided randomly on which of the routine works which of the three mutation operators will be performed or which projects will be affected by the project mutation. If a mutation took place, then the original chromosome is replaced with the mutated one. Finally, the mutated chromosome is added to the new population.

Step 6: Add chromosomes to the new population

In order to improve efficiency and converge speed we implement elitism as well. The main idea of elitism is to preserve the best genetic material by copying the best member of each generation into subsequent generations. Thus, before any new chromosomes are added to the new population, a fixed number of chromosomes with the highest fitness are copied from the old population to the new population. The Steps 3 - 6 are repeated until the new chromosome population size is equal to the size of the initial population.

Step 7: Replace the old population with the new population

Finally, in Step 7 the current chromosome population is replaced with the new population and the Steps 2 - 7 are repeated until the maximum number of iterations is reached.

3.3 Memetic Algorithm (MA)

The memetic algorithm, as a variant of the genetic algorithm, is called a hybrid evolutionary technique (Burke and Smith (2000)), where the hybridization is realized by integrating the genetic algorithm with local search techniques. The memetic algorithm modifies the genes in order to get an offspring with a higher fitness. The modification of genes is accomplished by using local search on every produced new chromosome before adding it to the new population, so after Step 2 and Step 5 in Figure 1.

In this paper we implement three local search algorithms, namely the *Steepest Hill Climbing (SHC)*, *Simulated Annealing (SA)* and *Tabu Search (TS)*. These are the most common local search techniques in the literature and Burke and Smith (1997b), Burke and Smith (1999b) have used these kind of algorithms with success on problems alike the PMSP. Burke and Smith (1997b), Burke and Smith (2000) affirm that the memetic algorithm is less sensitive to the quality and diversity of the initial population than the genetic algorithm. In spite of this observation, we still perform a local search on the randomly created initial population before starting the evolutionary part of the algorithm. In the literature, Burke and Smith (1997b), Burke and Smith (2000), Burke and Smith (1999a) and Burke and Smith (1999b) do not apply this initial local search, but Radcliffe and Surry (1994) do add this step.

As in Burke and Smith (2000), Burke and Smith (1999a) and Burke and Smith (1999b), we also implement here an iterative heuristic (IH) using the three local search heuristics mentioned above. The iterative heuristic repeatedly applies the local search optimiser to randomly generated solutions for the same number of times that the equivalent memetic algorithm applies local search. Thus, IH gives us possibility to compare the memetic algorithm with the repeated application of the individual local search operators.

All local search methods start looking for better solutions in the so-called *neighborhood* of the current solution. The neighborhood of a solution is the set of solutions that can be obtained by applying a very small local change to it. We implement the neighborhood in the following way. For routine works each execution is first moved one position to the left and thereafter one position to the right, unless the stopping criteria is met. We have three stopping criteria's, namely if a violation in the planning cycle occurs or the end of the planning horizon is reached or another execution is encountered. For projects, the starting time is set to one time period earlier or later, unless this new starting time is not in the given (earliest allowed staring time, latest allowed staring time) interval. Each change in the chromosome implies a new neighbor. Because of performance reasons instead of saving the exact neighbor we save just the move needed to create it.

As mentioned before, in this paper we focus on three local search algorithms, namely on Steepest Hill Climbing (SHC), Simulated Annealing (SA) and Tabu Search (TS). These methods are described below.

3.3.1 Steepest Hill Climbing

The steepest hill climbing algorithm for the PMSP is comparable with the algorithm used in Moscato and Schaerf (1997). The following steps are performed:

1. Set the current solution.
2. Compute the difference in costs for all members of the neighborhood and choose the one that implies the highest cost reduction. If there are more than one solutions with the highest reduction, then one of them is chosen randomly.
3. If the best neighbor is better than the current solution then return to step 1. If the original solution is better than all its neighbors, the local optimum is found and the algorithm terminates as well.

3.3.2 Simulated Annealing

The simulated annealing algorithm used in this paper is identical with the one from Moscato and Schaerf (1997) and Wolsey (1998). The main steps of the algorithm are described below.

1. Set the current solution.
2. Check whether the current temperature T is higher than the temperature limit. If not, then stop.

3. Given the current solution, choose randomly one of its neighbors.

4. Compute the difference in costs between the current solution and the selected neighbor. If the neighbor is better than the current solution, then the neighbor is accepted. Otherwise, the neighbor is accepted with a probability equal to $e^{-\Delta/T}$, where Δ is the difference in costs.

5. If the number of iterations for the current temperature has been reached, then the current temperature is decreased by the cooling rate α. Thus, $T_n = T_{n-1} \cdot \alpha$, where $0.8 < \alpha < 1$. Return to step 2.

3.3.3 Tabu Search

The basic principle of tabu search is to pursue the search whenever a local optimum is encountered by allowing non-improving moves. Cycling back to previously visited solutions is prevented by a tabu list, that records the recent history of the search. There are three parameters used for the tabu search, namely the tabu list length, the number of neighbors at each iteration and the total number of iterations.

The tabu search algorithm implemented for the PMSP can be described as follows:

1. Given the current solution, initialize an empty tabu list of a given length

2. Test whether the total number of neighbors is not smaller than the given number of neighbors to visit. If so, set the number of neighbors to visit equal to the total number of neighbors for this iteration of the algorithm.

3. For the number of neighbors to visit:

   a) Pick a random neighbor that has not been visited yet.

   b) Set the status of the picked neighbor to visited.

   c) Compute the costs difference for this neighbor.

   d) If this neighbor does not imply a cost reduction, check its tabu status.

   e) If this neighbor is not tabu or does improve the original solution and it is the best neighbor found so far, set this as the current best neighbor.

4. Update the tabu list. If the list is full, replace the oldest entry by this new one.

5. Create a neighborhood for the picked solution and return to step 1, unless the given maximum number of iterations has been reached.

For a complete detailed description of the tabu search algorithm we refer to Moscato and Schaerf (1997) and De Ree (2006).

3.4 Two-phase opportunities based heuristic (GA_OPP)

Since in Budai *et al.* (2006) the Opportunity Based Heuristic (OBH) gave in general good results for solving PMSP, we think that it is worth to use the idea of performing preventive maintenance works at opportunities in combination with the genetic algorithm. Therefore, in this paper a third heuristic approach, called GA_OPP, has been developed to solve the PMSP. GA_OPP is a two phase method that is based on opportunities. In the first phase, opportunities are created using the genetic algorithms and in the second phase all the executions of the preventive works are fitted as much as possible to these opportunities.

In the GA_OPP the chromosomes are represented as one dimensional binary arrays, where the columns represent the time periods. Thus, the j-th element indicates whether at period j there is an opportunity. Because the chromosomes contain all the opportunities, we will also call them as opportunity lists. To each chromosome a two dimensional array is assigned, which is used once the maintenance activities are fit to the opportunity lists. In this array the resulting maintenance planning is created, so an element on position (i, j) equal to 1 means that routine work i is planned to be executed in time period j.

The GA_OPP developed in this paper follows the steps presented in Figure 1. Some of the steps are exactly the same as in Section 3.2 (*e.g.* selection scheme), but others are completely different (*e.g.* generation of the initial population, crossover and mutation operators). Each step used for GA_OPP is explained in details below.

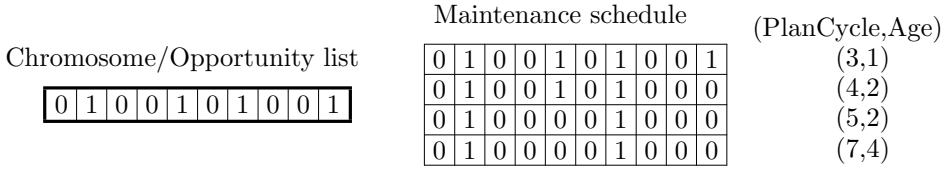

Figure 6: Example: chromosome and the attached maintenance schedule for GA_OPP

Step 1: Create initial population

The initial population is created randomly, but with the following restrictions:

1. Check beforehand the minimum time between two consecutive executions of the preventive routine maintenance activities (T_{min}), *i.e.* the smallest planning cycle.
2. Calculate the minimum of the difference between planning cycle and age over all activities. This is denoted by ${T^1}_{min}$.
3. Order the routine maintenance works such that their planning cycles are in increasing order.
4. Until the population size is reached, do the following steps:
   a) Draw $\lceil \frac{|T|}{T_{min}} \rceil$ random numbers in the $[1, |T|]$ interval, where $|T|$ is the planning horizon. A random number j means that at time period j there is an opportunity, so the j-th element of the chromosome becomes 1.
   b) Check whether the gap between the first opportunity and the beginning of the time period is smaller than ${T^1}_{min}$. If not, then at time period ${T^1}_{min} - 1$ a new opportunity is created.
   c) Check whether all the gaps between two consecutive opportunities of the chromosome are smaller than T_{min}. If not, then for each gap larger than T_{min} a new opportunity is repeatedly created at $T_{min} - 1$ periods from the beginning of the gap.
5. Until the population size is reached, the preventive maintenance activities are fit to each chromosome in the following way:
   a) for each routine maintenance activity and according to its planning cycle choose the last allowed opportunity in the opportunity list.
   b) plan each project based on its own duration and starting time, such that the already existing opportunities are used as much as possible. If for fitting the projects new opportunities are needed, then these new opportunities are added to the opportunity list.

As the example in Figure 6 shows, in the maintenance schedule only the opportunities from the chromosome are used as possessions.

Step 2: Fitness evaluation

For GA_OPP the fitness of a chromosome is equal to the fitness of the attached maintenance schedule and that is calculated in the same way as we have already described in Section 3.2 for GA.

Step 3: Selection scheme

The selection schemes for the GA_OPP are identical with those from Section 3.2.

Step 4: Crossover

The *one point crossover* operator (see Figure 7) starts by choosing a random number between 1 and the length of planning horizon. Based on this crossover point the two chosen chromosomes are divided in two parts. For creating the two offsprings we interchange the opportunity lists of the two parents before and after this random crossover point. This might result in not feasible offsprings, since gaps between two consecutive opportunities might be longer than T_{min} or the gap between the first opportunity and the

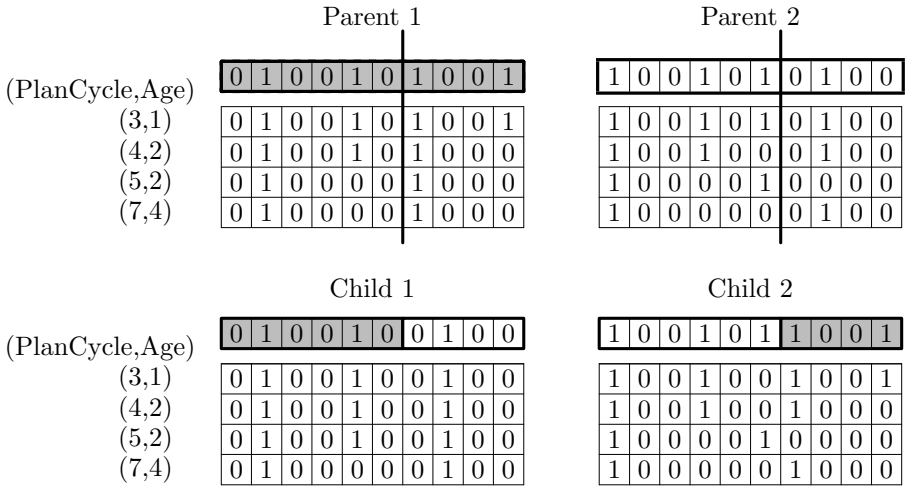

Figure 7: Crossover for GA_OPP

beginning of the time period is smaller than $T^1{}_{min}$. If such gaps are detected, then new opportunities are added to the opportunity list based on the Steps 4*b* and 4*c* used in creation of the initial population for GA_OPP. Finally, the maintenance works are fitted to the newly created two offsprings.

Step 5: Mutation

Beside crossover, mutation of the chromosomes is another way to create offsprings. For GA_OPP we use four types of mutation operators, namely *remove excess opportunities*, *insert opportunities*, *postpone executions* and *remove excess executions*. The first mutation operator might change the chromosomes, but not the maintenance schedule attached to the chromosomes, while the last three operators might change the maintenance schedule too. Actually, the last two operators make first possible changes in the maintenance planning attached to the chromosome and after that the chromosome will be updated, too. Below we describe each of these operators in details.

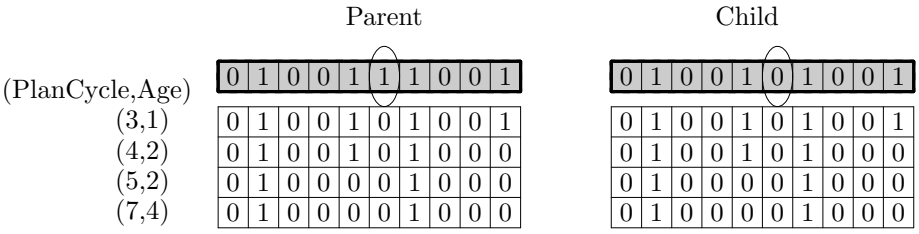

Figure 8: Remove excess opportunities mutation

The *remove excess opportunities operator* (Figure 8) deletes all the opportunities from the chromosome that have not been used for the maintenance planning.

With the *insert opportunities operator* (Figure 9) new opportunities are added to the chromosome. For each element of the chromosome and based on a user defined probability value it is decided whether that element will be changed into a new opportunity (unless it is already an opportunity) or not. Thereafter, the maintenance activities are fit to the chromosome.

The *postpone executions operator* (Figure 10) checks first in the maintenance schedule whether there exist possessions for which only one maintenance work is planned. If there are such of possessions, then it is tried for each of these possessions to postpone the single executions to a next opportunity, unless the stopping criteria is met. If executions could be postponed, then this might result in unused opportunities. Therefore, the chromosome will be updated as well.

Finally, the *remove excess executions operator* (Figure 11) checks in the maintenance schedule for each routine work whether there are excess executions and if so, then these executions are deleted. A given

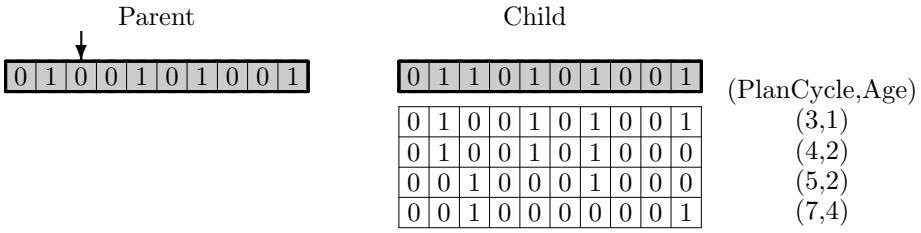

Figure 9: Insert opportunities mutation

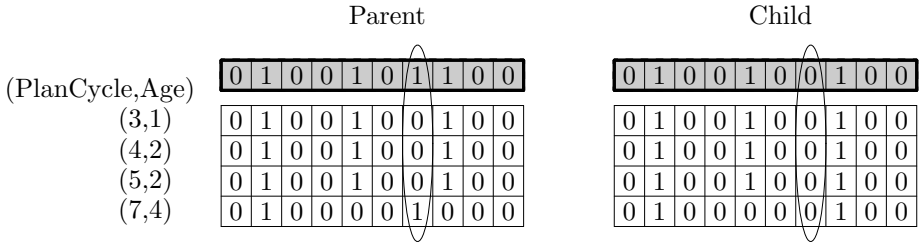

Figure 10: Postpone executions mutation

execution of a routine work is called excess execution, and thus it can be removed, if the time period between the next execution and the previous execution of the same work is smaller then the planning cycle. Removing executions might result in useless opportunities, therefore in the end the chromosome will be updated, too.

The crossover and four mutation operators are performed with user defined probabilities. First two parents are selected from the current population and after that it is determined whether crossover will be performed on these parents. If so, these two parents are replaced by their two children. After that, either both children or both parents (if the crossover was not selected) will be mutated individually. Based on the user defined probability value it is determined which of the four mutation operators will be used to generate new offsprings. If a mutation took place, then the original chromosome (and the original maintenance schedule) is replaced with the mutated one. Finally, the mutated chromosome is added to the new population.

Step 6: Add chromosomes to the new population

For GA_OPP heuristic the elitism is implemented, too. This step is identical with Step 6 described in Section 3.2 for GA.

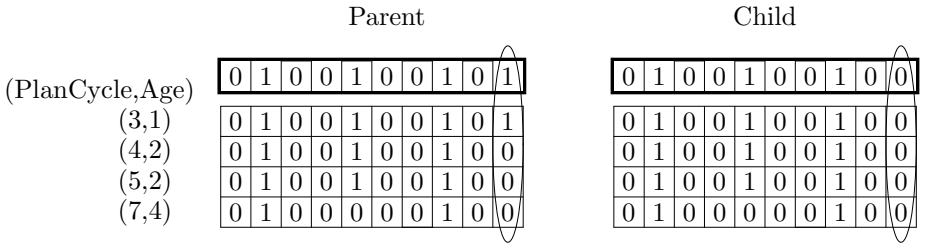

Figure 11: Remove excess executions mutation

Step 7: Replace the old population with the new population

In Step 7 the current chromosome population is replaced with the new population and Steps 2 - 7 are repeated until the maximum number of iterations is reached.

4 Computational results

Data

In this section we present the results of solving PMSP using the genetic algorithm, memetic algorithm, iterative heuristic and the two-phase opportunities based heuristic. To be able to compare these results with the solution found in Budai *et al.* (2006) by the exact model and the greedy heuristics, we use the same problem instances and scenarios as in the referenced article.

We recall, that these are twenty randomly created problem instances, ten instances with 15 and ten instances with 25 maintenance works. The models and heuristics are tested for two scenarios and for two different possession costs ($PossC = 25$ and $PossC = 75$). In the first scenario each routine work can be combined with all other routine works and projects, but projects cannot be combined with other projects. In the second scenario it is assumed that for one particular group of two works and one particular group of three works, the works within these groups cannot be carried out at the same time. The works within such a group are randomly chosen.

The planning horizon for the generated instances is 2 years and the discrete time periods are weeks. We assume that the track possession cost is the same for each week within the planning horizon. Furthermore, we assume that each routine maintenance work has different planning cycles, ages, maintenance costs and each project has a different duration, possible earliest and latest starting time. All the tests are executed on a Pentium IV 2.5GHz 261MB RAM.

Parameter settings

In GA, MA, IH and GA_OPP there are a couple of parameters that need to be set by the user. These are not only the probability values for crossover or mutation, but also parameters as population size, elite size, number of iterations, *etc.*

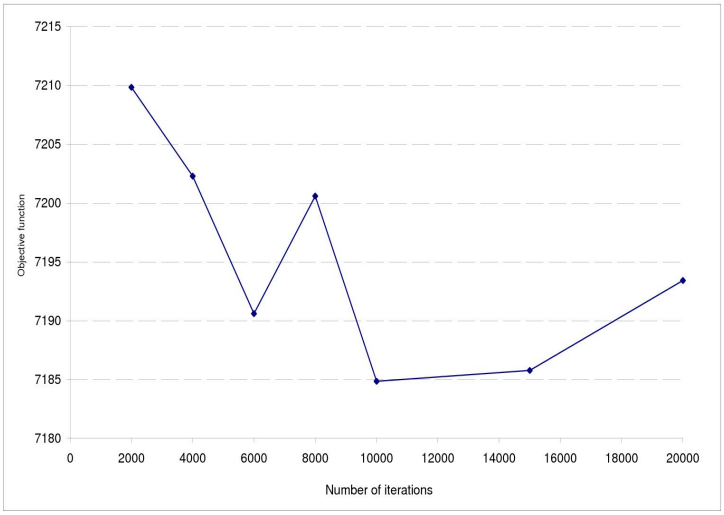

Figure 12: Results of GA for different number of iterations

The parameters used for GA, MA, IH and GA_OPP are determined after several experiments. One of the experiments that we carried out is to find out the right number of iterations for the GA. Figure 12 shows the results of different runs with different number of iterations. It turns out that number

of iterations equal to 10000 results in the lowest objective value. In order to be able to compare the results for GA with the GA_OPP we decided to use the same number of iterations for GA_OPP, too. Moreover, for the GA and GA_OPP the population size and the elite size have been chosen to 100 and 2, respectively, based on a range of experiments. Furthermore, the rank based roulette wheel proved to be the best selection method.

Parameters	GA	GA_OPP	MA/IH		
			SHC	SA	TS
Nr. iterations	10000	10000	2000	2000	2000
Population size	100	100	100	100	100
Elite size	2	2	2	2	2
Selection scheme type	rank based roulette wheel	rank based roulette wheel			
Prob. crossover	0.8	0.8	0.8	0.8	0.8
Prob. zero insertion	0.001		0.001	0.001	0.001
Prob. zero removal	0.001		0.001	0.001	0.001
Prob. shift single one	0.05		0.05	0.05	0.05
Prob. project mutation	0.001		0.001	0.001	0.001
Prob. join together columns	0.001		0.001	0.001	0.001
Prob. opportunity insertion		0.01			
Prob. remove unused opportunities		0.8			
Prob. postpone executions		0.8			
Prob. remove excess executions		0.8			
Starting temperature				2	
Cooling rate				0.9	
Nr. iterations per temperature				10	
Temperature limit				0.4	
Tabu list length					10
Size of subset of neighbors					15
Tabu search nr. of iterations					15

Table 1: Parameters used for GA, GA_OPP, MA and IH (SHC, Steepest Hill Climbing; SA, Simulated Annealing; TS, Tabu Search)

The probability of crossover has been chosen for GA and GA_OPP equal to 0.8. For GA the probability of zero insertion, zero removal, project mutation and join together columns mutation operators are set to 0.001 and the probability of shifting a single execution to 0.05. The probability of mutation for GA_OPP are chosen slightly higher than for GA, namely the probability of removing unused opportunities, postponing executions and removing excess executions are set to 0.8 and the probability of inserting opportunities to 0.01.

The number of iterations for the memetic algorithm using the three types of local search optimizers is chosen equal to 2000. For the tabu search the length of the tabu list is set to 10, the number of iterations and the size of subset of neighbors for the tabu search is set to 15. Simulated annealing has only one tunable parameter, which is the cooling rate. Based on different empirical tests the cooling rate is fixed to 0.9. The initial temperature is set to 2 and as the algorithm progresses this temperature reduces gradually. In total 15 temperature levels are visited, resulting in a temperature limit of 0.4. For each temperature 10 runs are performed. All these parameter values were identified to be reasonable for the test problems.

All the parameters used for GA, GA_OPP, MA and IH are summarized in Table 1. These parameters are found to be those that gave the best results for GA, GA_OPP, MA and IH.

Analysis of the results

The exact method (PMSP) and the two greedy heuristics (OBH - Opportunity Based Heuristic and MCWF - Most Costly Work First) from Budai *et al.* (2006) are compared here with the 8 new algorithms, namely with the genetic algorithm (GA), with the memetic algorithm using three local search algorithms: tabu search (MA(TS)), simulated annealing (MA(SA)) and steepest hill climbing (MA(SHC)), with the iterative heuristic using tabu search (IH(TS)), simulated annealing (IH(SA)) and steepest hill climbing (IH(SHC)) and finally with the two-phase opportunities based heuristic (GA_OPP). These 8 algorithms are tested on all the 80 problem instances. The best solution for a given problem instance is chosen from the solutions of 10 runs of the algorithm on this problem instance.

Method	Overall best	Best heuristic
CPlex	22 (14 opt)	-
OBH	0	1
MCWF	0	1
GA	0	0
GA_OPP	40 (3 opt)	51 (3 opt)
MA(TS)	15	16
MA(SA)	7 (2 opt)	10 (2 opt)
MA(SHC)	3 (2 opt)	3 (2 opt)
IH(TS)	0	0
IH(SA)	0	0
IH(SHC)	0	0

Table 2: Number of times a method proofs to be the best overall method or the best heuristic

Table 2 shows that the CPlex solver returns a solution that is not improved by any other method in 22 of the 80 cases. However, for 3 instances the GA_OPP and for 2 instances both MA(SA) and MA(SHC) find the same (optimal) solutions as the CPlex solver finds. We recall that the exact method (CPlex) could find only in 14 out of 80 instances an optimal solution within 3 hours. Since the CPlex solver was stopped after 3h, the CPlex solutions might be improved by any of these new heuristics, which is actually the case for 72.5% of the instances.

If we exclude the exact model from the comparison then for 63% of the instances GA_OPP gives the best solution, outperforming far the rest of the solution methods. Moreover, integrating the genetic algorithm with local search techniques gives better results than just simply performing genetic algorithm. As Table 2 shows, in any of the 80 cases the genetic algorithm and the three iterative methods do not return a best solution. The three memetic algorithms return however good results, the memetic algorithm with the tabu search outperforming the MA(SA) and MA(SHC), as Burke and Smith (1997b) and Burke and Smith (2000) conclude too. However, the MA(SA) and MA(SHC) find for 2 instances the optimal solutions.

	Method	Method performs worse										
		CPlex	OBH	MCWF	GA	GA_OPP	MA (TS)	MA (SA)	MA (SHC)	IH (TS)	IH (SA)	IH (SHC)
Method performs better	CPlex	-	70	79	54	22	39	35	39	80	80	80
	OBH	10	-	70	11	2	3	4	5	80	80	80
	MCWF	1	10	-	3	1	3	3	3	58	57	58
	GA	26	69	77	-	5	1	1	7	80	80	80
	GA_OPP	58	78	79	75	-	59	61	68	80	80	80
	MA(TS)	41	77	77	79	21	-	31	45	80	80	80
	MA(SA)	45	76	77	79	19	49	-	56	80	80	80
	MA(SHC)	41	45	77	73	12	35	24	-	80	80	80
	IH(TS)	0	0	22	0	0	0	0	0	-	40	45
	IH(SA)	0	0	23	0	0	0	0	0	40	-	43
	IH(SHC)	0	0	22	0	0	0	0	0	35	37	-

Table 3: Comparison of pairs of methods used for solving PMSP

In Table 3 we compare pairs of methods used for solving PMSP, because this provides good insight into the quality of the algorithms. It seems that GA performs better than OBH and MCWF, for 26 instances it gives even better results than the exact method. Furthermore, GA outperforms GA_OPP only for 5 instances and for each of the 80 instances the iterative heuristic. Actually, all the three iterative heuristics perform very poor, IH(SHC) being the worst out of these three methods. Comparing in Table 3 the performance of MA(TS), MA(SA) and MA(SHC), we can conclude that MA(SA) is equally good as MA(TS), but slightly better than MA(SHC).

In Table 4 and Table 5 the average results for all methods is given for each scenario separately, for different number of routine works and possession costs. Table 4 summarizes the results for the scenario 1 and Table 5 for scenario 2. In these tables we present the averages of the least costs, the computation times for 10 problem instances and the relative differences between the exact solution and the solution found by the heuristics.

Method	PossC	SCENARIO 1					
		n = 25			n = 15		
		VOpt/ObjV	TCPU(s)	RdOH(%)	VOpt/ObjV	TCPU(s)	RdOH(%)
CPlex	25	10036.37*	>10800	-	5681.4	9455.84	-
OBH	25	10085.22	1.4	0.49	5771.43	1.4	1.58
MCWF	25	10410.24	1.5	3.73	6041.28	1.1	6.33
GA	25	9977.44	1020	-0.59	5694.52	1020	0.23
GA_OPP	25	9858.61	2760	**-1.77**	5611.87	1800	**-1.22**
MA(TS)	25	9893.81	1800	-1.42	5643.36	1200	-0.67
MA(SA)	25	9890.24	1980	-1.46	5640.46	1260	-0.72
MA(SHC)	25	9913.29	1560	-1.23	5645.29	780	-0.64
IH(TS)	25	11027.73	1380	9.88	6332.83	900	11.47
IH(SA)	25	11032.09	2580	9.92	6336.83	1440	11.54
IH(SHC)	25	11033.63	4740	9.94	6334.97	1680	11.50
CPlex	75	11574.71*	>10800	-	6720.74	7424.45	-
OBH	75	12365.91	1.4	6.84	7201.67	1.2	7.16
MCWF	75	12361.87	1.2	6.8	7524.28	1.2	11.96
GA	75	11947.99	1500	3.22	7045.72	1020	4.84
GA_OPP	75	11323.48	2760	**-2.17**	6688.93	1740	**-0.47**
MA(TS)	75	11714.65	1800	1.21	6923.67	1200	3.02
MA(SA)	75	11669.31	2040	0.82	6890.61	1260	2.53
MA(SHC)	75	11690.16	1560	1	6944.76	840	3.33
IH(TS)	75	14542.56	4740	25.64	8815.91	900	31.17
IH(SA)	75	14545.82	2520	25.67	8825.00	1440	31.31
IH(SHC)	75	14549.54	4740	25.70	8832.98	1740	31.43

Table 4: Computational results for PMSP - Scenario 1 (* - No optimal solution could be found within 3h. VOpt, optimal value of PMSP or the best value found within 3h; ObjV, solution value of the heuristic; TCPU, CPU time; RdOH, relative difference between VOpt and ObjV)

GA, GA_OPP and all three memetic algorithms improve on average the best solution found by the CPlex solver for both scenarios and possession cost equal to 25. However if the possession cost equals 75, then only GA_OPP can improve the solutions found by the CPlex solver. The CPU time of the heuristics vary from 3 to 50 minutes, but anyway far under the 3 hours that was needed on average by the CPlex solver. From Table 4 and Table 5 we conclude again that GA_OPP gives on average the best solution, followed by the MA(SA) and MA(TS).

Comparing the results of the memetic algorithms with the three local search algorithms, we observe that on average the MA(SA) has the highest computation time, followed by MA(TS), while the MA(SHC) uses the least time. Measured over both scenarios, the memetic algorithm with tabu search improves the results of the genetic algorithm with 1.38% on average on the cost of about 13% extra computational time. For the memetic algorithm with the simulated annealing these percentages are 1.6% and 24.6% respectively, while the memetic algorithm with steepest hill climbing improves the results of the genetic algorithm with about 1.2% whilst reducing computation time with 10%.

The iterative heuristics are for both scenarios among the slowest methods, especially if the number of works increases. The three iterative heuristics give very comparable results, implying that when performed on a random initial solution, the used local search optimiser does not make a large difference. The differences between the different memetic algorithms are slightly bigger than between the different iterative heuristics. Note that the iterative heuristic only faces randomly generated solutions, which can be compared with the initialization phase of the memetic algorithms. Therefore, the better performance of the memetic algorithms in comparison with the iterative heuristics are probably caused by the genetic part of the memetic algorithm. Anyway, the results with respect to the iterative heuristics lead us to conclude that these algorithms are not suitable for solving PMSP. Burke and Smith (1999b), Burke and Smith (1999a), however report that the iterative heuristics (IH(TS), IH(SA) and IH(SHC)) provide reasonably good solutions for their maintenance scheduling problems, which is not anymore the case in Burke and Smith (2000). Nevertheless, all these articles clearly state that the memetic algorithm using tabu search as a local optimiser produces the best results in comparison with the GA, MA(SA) and MA(SHC).

In order to reduce the total possession cost and the inconvenience for the train operators and travelers, the maintenance works that do not exclude each other should be scheduled as much as possible together for the same time period. However, grouping mostly implies that one deviates from the originally planned execution moments and thus some maintenance actions are more often performed than originally planned. This involves costs as well. In Table 6 we present for different solution methods and averaged over 10 instances the gains and losses achieved by combining different maintenance activities with each other.

Method	PossC	SCENARIO 2					
		n = 25			n = 15		
		VOpt/ObjV	TCPU(s)	RdOH(%)	VOpt/ObjV	TCPU(s)	RdOH(%)
CPlex	25	10086.84*	>10800	-	5705.28	8462.13	-
OBH	25	10132.89	1.3	0.46	5816.55	1.3	1.95
MCWF	25	10283.31	1.6	1.95	6030.27	1	5.7
GA	25	10037.4	1440	-0.49	5702.92	900	-0.04
GA_OPP	25	9932.88	3000	**-1.53**	5680.4	1980	-0.44
MA(TS)	25	9940.42	1320	-1.45	5662.87	900	**-0.74**
MA(SA)	25	9938.75	1560	-1.47	5663.5	900	-0.73
MA(SHC)	25	9963.29	1140	-1.22	5688.35	600	-0.3
IH(TS)	25	11041.51	1380	9.46	6322.09	900	10.81
IH(SA)	25	11045.73	2640	9.51	6325.09	1500	10.86
IH(SHC)	25	11042.56	4740	9.47	6331.05	1920	10.97
CPlex	75	11710.15*	>10800	-	6889.13	7163.2	-
OBH	75	12505.64	1.4	6.79	7359.05	1.4	6.82
MCWF	75	12363.12	1.2	5.58	7811.35	1.5	13.39
GA	75	12052.14	1440	2.92	7104.31	900	3.12
GA_OPP	75	11607.02	3060	**-0.88**	6922.39	1980	0.48
MA(TS)	75	11841.97	1320	1.13	6980.62	900	1.33
MA(SA)	75	11811.56	1620	0.87	6959.36	900	1.02
MA(SHC)	75	11868.36	1200	1.35	7006.41	600	1.7
IH(TS)	75	14551.14	1380	24.26	8781.47	900	27.47
IH(SA)	75	14576.5	2640	24.48	8797.93	1500	27.71
IH(SHC)	75	14565.59	4740	24.38	8793.33	1920	27.64

Table 5: Computational results for PMSP - Scenario 2

Method	PossC	n = 25		n = 15	
		Loss	Gain	Loss	Gain
GA	25	282.1	2822.5	177.8	1460
GA_OPP	25	248.3	2925	222.7	1602.5
MA(TS)	25	251	2875	166.7	1482.5
MA(SA)	25	257.4	2872.5	192.8	1485
MA(SHC)	25	275.4	2867.5	188.6	1522.5
GA	75	407.6	8767.5	276.5	4672.5
GA_OPP	75	645.6	10117.5	377.2	5332.5
MA(TS)	75	384.3	8955	267.1	4717.5
MA(SA)	75	410.0	9000	293.9	4807.5
MA(SHC)	75	442.3	9097.5	280.6	4755

Table 6: Gains/losses achieved by clustering maintenance activities

Without loss of generality we consider here only the routine works and we assume that each routine work can be combined with all the other routine works. The losses are calculated as function of the maintenance costs and the number of time periods that the executions of a routine works are earlier planned than their cycle length requires. The gains are equal to the possession cost when all the routine works are performed separately minus the possession cost under the policy. Once the possession cost per time period rises from 25 to 75, the losses increase as well. The reason behind this is that the more expensive the possession per time period is, the more works are planned for the same time period. Thus, the executions of the works are planned much earlier than the cycle length requires. The losses incurred for too early executions of the maintenance work in order to combine works together is reduced from 14% of the reductions in the total possession cost to just 5%.

In Figure 15 we present an example for a maintenance schedule made by the GA_OPP, that contains 15 routine works and 2 projects and with a planning horizon of 104 weeks (because of lack of space only 75 weeks are shown). All the routine works are allowed to be combined with other routine works or projects, but the projects cannot be combined with other projects. In the first 15 rows the routine works and in the last 2 rows the projects are scheduled. In order to minimize the possession costs the maintenance works are scheduled as much as possible together in the same time period. The Figure 15 clearly shows that the groups of maintenance works are not fixed during the planning period.

Furthermore, in Figure 13 and Figure 14 we compare the convergence of the GA with the GA_OPP and the convergence of MA(TS) with MA(SA) and MA(SHC), respectively, for the same problem instance that has been used for Figure 15. Comparing the convergence patterns of GA and GA_OPP in Figure 13, we observe that the initial solution is improved rapidly in the first hundreds of iterations. In the case of GA a major improvement in the solution value occurs approximately at iteration 3000 after which further

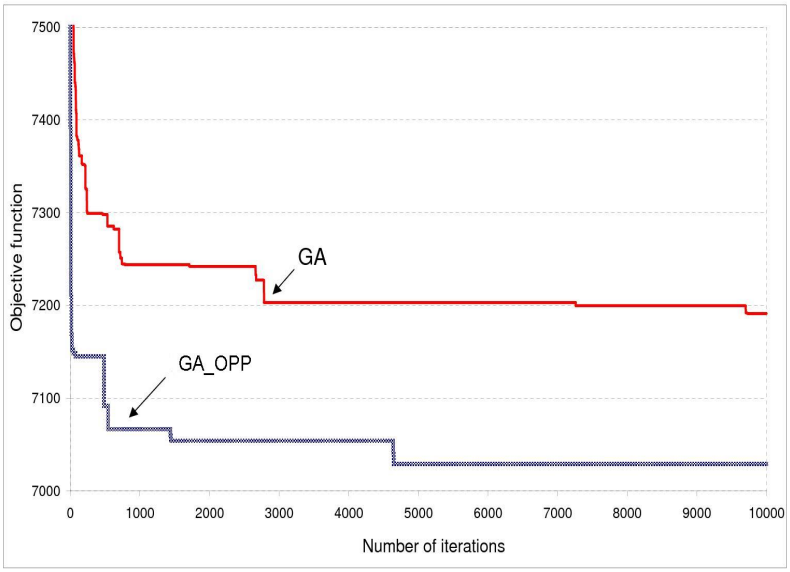

Figure 13: Progress of the GA and GA_OPP based on the best costs per iteration

improvements are marginal. In the case of GA_OPP the last major improvement occurs around iteration 4500. The solution does not change behind this point. In Figure 14 the initial solution is improved fast before iteration 200 and from that point very little improvements in the solution value can be noticed. From the solutions found by the three memetic algorithms we can say the MA(SA) outperforms the MA(TS) which performers slightly better than SA(SHC). The best solution found by MA(SA) is however significantly improved by the GA_OPP.

In conclusion, we can say that the solution methods developed in this paper, with the exception of the iterative heuristics, improved substantially the greedy heuristics from Budai *et al.* (2006). Moreover, on average GA_OPP improved the solution found by the CPlex solver for both scenarios and possession costs.

5 Conclusions

In this paper our objective was to develop better techniques than the greedy heuristics (OBH and MCWF) developed in Budai *et al.* (2006) to solve the preventive maintenance scheduling problem (PMSP). We recall, that PMSP aims to assign (short) repetitive routine maintenance works and (long) unique projects to different time periods (months/weeks), minimizing the track possession cost and the maintenance cost.

Since in the literature several authors recommend using meta-heuristics for maintenance scheduling problems comparable to PMSP, in this paper we implemented the genetic algorithm, memetic algorithm using three local search techniques (tabu search, simulated annealing and steepest hill climbing) and the iterative heuristic using the same local search techniques as the memetic algorithm for solving the PMSP. Moreover, we developed for the PMSP a two-phase opportunities based heuristic (GA_OPP), where in the first phase, opportunities are created by using the GA and in the second phase all the executions of the preventive works are fitted as much as possible to these opportunities.

The genetic algorithm and the memetic algorithms with the three local search algorithms improve on average the solutions found by the CPlex solver for the instances where the possession cost is low (PossC = 25), while for high possession cost (PossC = 75) the solution found by the CPlex solver cannot be improved anymore, resulting in a relative difference between the solution value of the exact method and the solution value of the heuristic of 0.8% to 4.8%. Comparing the performance of the genetic algorithms with the performance of the memetic algorithms using the tabu search, simulated annealing and steepest hill climbing optimizers, we can conclude that the local search optimizers improve the genetic algorithm

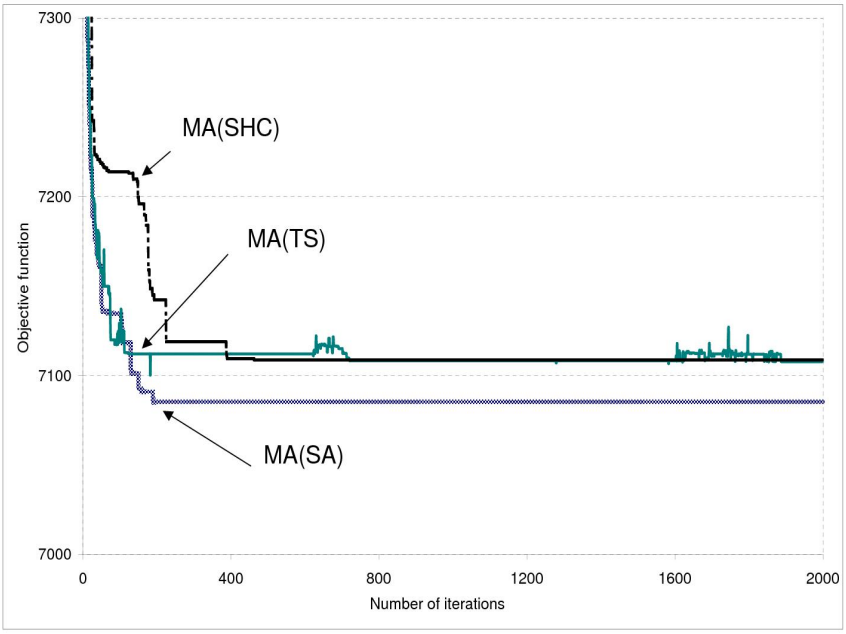

Figure 14: Progress of the memetic algorithm with the tabu search, simulated annealing and steepest hill climbing optimizers based on the best costs per iteration

for almost all of the 80 instances. Within the memetic algorithm, the simulated annealing algorithm tends to give the best results in general, followed by the tabu search and finally by steepest hill climbing. The iterative heuristic can be used to make comparisons, but it is not advisable to be used as a method to solve the PMSP, since it provides very poor results. The performance of the memetic algorithm is mainly caused by the genetic part, while the local search optimizers cause the improvement with respect to the genetic algorithm itself.

Furthermore, from the computational results we can conclude that GA_OPP can be used very well to solve PMSP, since on average GA_OPP improved the solution found by the CPlex solver for both scenarios and possession costs. Moreover, on average the GA_OPP outperformed the memetic algorithms with the three local search optimizers for 78% of the instances and the genetic algorithm for 94% of the instances, too. The success of the GA_OPP is due to the fact that it uses information about the structure of the problem. Given the opportunities the problem is much easier to solve. It also shows that standard GA's have problems in finding good solutions if only few paths in the solution space are possible. Furthermore, the solution value of the OBH and MCWF is on average significantly improved by the GA, GA_OPP and MAs, but not by the iterative heuristics. Comparing the computational times of these algorithms, we can conclude that OBH and MCWF are still the fastest heuristics, followed by the genetic and memetic algorithms. The GA_OPP needs somewhat more time than the GAs and MAs, but on average its CPU time (29-52 minutes) is still far below the computational time that the CPlex solver needed.

The ideas behind GA_OPP can also be extended to the maintenance planning on a network, especially the decomposition ideas. On a network level there is interest in whether adjacent single track grids are maintained at the same or adjacent time moments and whether there is a passage possible within the network. This can be taken into account when generating the opportunity structure by defining an appropriate cost structure. Fitting in the precise maintenance activities can be done on a second level in almost the same way.

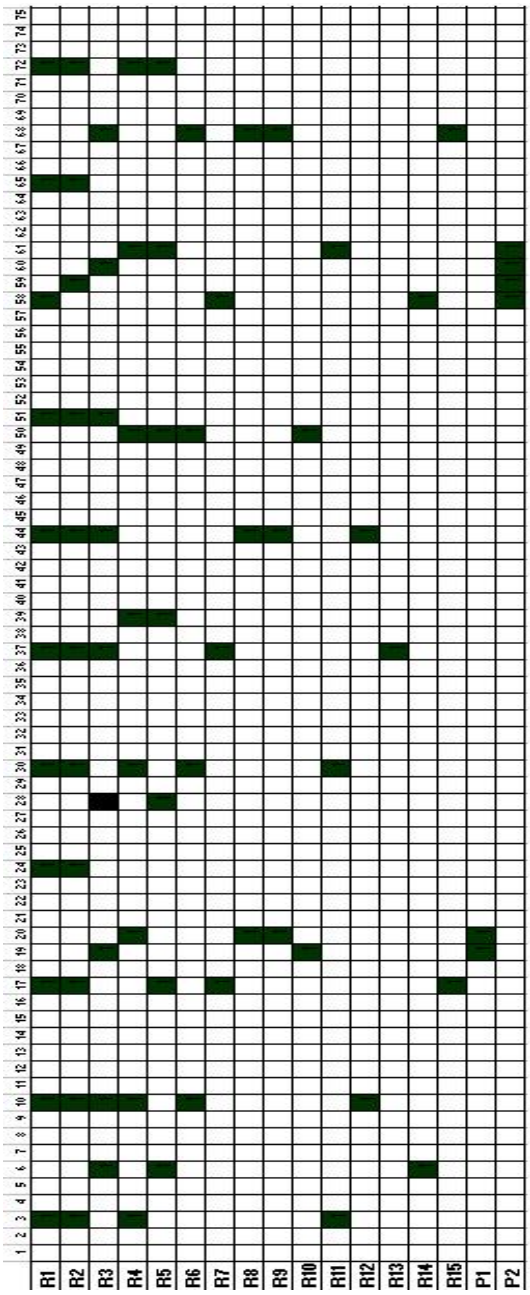

Figure 15: Example for a maintenance schedule containing 15 routine works and 2 projects

Acknowledgements

The authors would like to thank Diederik de Ree and Joel van 't Wout for their work in the preliminary phase of this article.

References

Abdulwhab, A., R. Billinton, A. Eldamaty, and S. Faried (2004), Maintenance Scheduling Optimization Using a Genetic Algorithm (GA) with a Probabilistic Fitness Function, *Electric Power Components and Systems*, **32**, 1239–1254.

Budai, G. and R. Dekker (2002), An Overview of Techniques Used in Planning Railway Infrastructure Maintenance, in W. Geraerds and D. Sherwin (eds.), *Proceedings of IFRIMmmm (maintenance management and modelling) conference, Växjö, Sweden*, Växjö University, Sweden.

Budai, G., D. Huisman, and R. Dekker (2006), Scheduling Preventive Railway Maintenance Activities, *Journal of the Operational Research Society*, **57**, 1035–1044.

Burke, E., J. Clark, and A. Smith (1997a), Four Methods for Maintenance Scheduling, in *Proceeding of the International Conference on Artificial Neural Networks and Genetic Algorithms*, Springer.

Burke, E. and A. Smith (1997b), A Memetic Algorithm for the Maintenance Scheduling Problem, in *Proceedings of the ICONIP Conference, Dunedin, New Zealand*.

Burke, E. and A. Smith (1999a), A Memetic Algorithm to Schedule Planned Grid Maintenance, in *Proceedings of the International Conference on Computational Intelligence for Modelling Control and Automation*, IOS Press.

Burke, E. and A. Smith (1999b), A Memetic Algorithm to Schedule Planned Maintenance for the National Grid, *The ACM Journal of Experimental Algorithmics*, **4**, 1–13.

Burke, E. and A. Smith (2000), Hybrid Evolutionary Techniques for the Maintenance Scheduling Problem, in *IEEE Transactions on Power Systems*, vol. 15.

Chan, W., T. Fwa, and K. Hoque (2001), Constraint Handling Methods in Pavement Maintenance Programming, *Transportation Research Part C*, **9**, 175–190.

Cheung, B., K. Chow, L. Hui, and A. Yong (1999), Railway Track Possession Assignment Using Constraint Satisfaction, *Engineering Applications of AI*, **12**, 599–611.

Dahal, K. and N. Chakpitak (2007), Generator Maintenance Scheduling in Power Systems Using Metaheuristic-based Hybrid Approaches, *Electric Power Sytems Research*, **77**, 771–779.

De Ree, D. (2006), Scheduling Preventive Railway Maintenance Using Memetic Algorithms, Econometrics and Management Science, Erasmus University Rotterdam, Bachelor Thesis.

Dekker, R. and C. van Rijn (1996), *PROMPT - a Decision Support System for Opportunity Based Preventive Maintenance*, vol. 154, Springer-Verlag, Berlin, pp. 530–549.

Den Hertog, D., J. van Zante-de Fokkert, S. Sjamaar, and R. Beusmans (2005), Optimal Working Zone Division for Safe Track Maintenance in The Netherlands, *Accident Analysis and Prevention*, **37**, 890–893.

Fwa, T., C. Tan, and W. Chan (1994), Road-maintenance Planning Using Genetic Algorithms II: Analysis, *Journal of Transportation Engineering, ASCE 120*, **5**, 710–722.

Grimes, C. (1995), Application of Genetic Techniques to the Planning of Railway Track Maintenance Work, in A. Zalzala (ed.), *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, Sheffield, UK*, Conference Publication 414, IEE, University of Sheffield.

Higgins, A. (1998), Scheduling of Railway Maintenance Activities and Crews, *Journal of the Operational Research Society*, **49**, 1026–1033.

Holland, J. (1975), *Adaption in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.

Improverail (2002), http://www.tis.pt/proj/improverail/Downloads/D6Final.pdf (last accessed on 30 November 2006).

Lapa, C., C. Pereira, and M. de Barros (2006), A Model for Preventive Maintenance Planning by Genetic Algorithms Based in Cost and Reliability, *Reliability Engineering and System Safety*, **91**, 233–240.

Lapa, C., C. Pereira, and A. Mol (2000), Maximization of a Nuclear System Availability Through Maintenance Scheduling Optimization Using a Genetic Algorithm, *Nuclear Engineering and Design*, **196**, 219–231.

Li, S., C. Ting, C. Lee, and S. Chen (2002), Maintenance Scheduling of Oil Storage Tanks Using Tabu-Based Genetic Algorithm, in *Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence*.

Liu, C., A. Hammad, and Y. Itoh (1997), Maintenance Strategy Optimization of Bridge Decks Using Genetic Algorithm, *Journal of Transportation Engineering*, **123**, 91–100.

Morcous, G. and Z. Lounis (2005), Maintenance Optimization of Infrastructure Networks Using Genetic Algorithms, *Automation in Construction*, **14**, 129–142.

Moscato, P. and A. Schaerf (1997), Local Search Techniques for Scheduling Problems - A Tutorial, unpublished paper, www.cs.bgu.ac.il/ cp041/Andrea_Local_Search.ps (last accessed on 17 January 2009).

Munoz, A., S. Martorell, and V. Serradell (1997), Genetic Algorithms in Optimizing Surveillance and Maintenance of Components, *Reliability Engineering and System Safety*, **57**, 107–120.

Negnevitsky, M. (2005), *Artificial Intelligence*, Pearson Education, Essex.

Negnevitsky, M. and G. Kelareva (1999), Genetic Algorithms for Maintenance Scheduling in Power Systems, in *Australasian Universities Power Engineering Conference and IEAust Electric Energy Conference, Darwin*.

Radcliffe, N. and P. Surry (1994), Formal Memetic Algorithms, in T. Fogarty (ed.), *Evolutionary Computing: AISB Workshop*, Springer-Verlag.

Roundy, R. (1985), 98%-Effective Integer-Ratio Lot-Sizing for One-Warehouse Multi-Retailer Systems, *Management Science*, **31**, 1416–1430.

Sastry, K., D. Goldberg, and G. Kendall (2005), Genetic Algorithms, in E. Burke and G. Kendall (eds.), *Search Methodologies - Introductory Tutorials in Optimization and Decision Support Techniques*, chap. 4, Springer US, pp. 97–125.

Savic, D., G. Walters, and J. Knezevic (1995a), Optimal Opportunistic Maintenance Policy Using Genetic Algorithms, 1: Formulation, *Journal of Quality in Maintenance Engineering*, **1**, 34–49.

Savic, D., G. Walters, and J. Knezevic (1995b), Optimal Opportunistic Maintenance Policy Using Genetic Algorithms, 2: Analysis, *Journal of Quality in Maintenance Engineering*, **1**, 25–34.

Sriskandarajah, C., A. Jardine, and C. Chan (1998), Maintenance Scheduling of Rolling Stock Using a Genetic Algorithm, *Journal of Operational Research Society*, **49**, 1130–1145.

Swier, J. (2003), Outsourcing in The Netherlands; Experiences and Developments, Presentation, http://www.promain.org/images/brussel/ProRail_ProMain_okt_2003_2.pdf (last accessed on 28 November 2008).

Van Zante-de Fokkert, J., D. den Hertog, F. van den Berg, and J. Verhoeven (2007), The Netherlands Schedules Track Maintenance to Improve Track Workers' Safety, *Interfaces*, **37**, 133–142.

Wolsey, L. (1998), *Integer Programming*, New York, Wiley - Interscience.