OPTIMIZATION OF LARGE SCALE MAINTENANCE NETWORKS

László Kota
Department of Materials Handling and Logistics
University of Miskolc
Hungary
altkota@uni-miskolc.hu

Károly Jármai
Department of Materials Handling and Logistics
University of Miskolc
Hungary
altjar@uni-miskolc.hu

ABSTRACT
This paper describes a comparison between a single phase evolutionary programming algorithm for the fixed destination multi-depot multiple travelling salesman problem with multiple tours problem (mmTSP) and the generalized tabu search algorithm. This optimization problem widely appears in the field of logistics mostly in connection with maintenance networks. In these types of systems, there are devices or sometimes whole sites. In this article they called objects and handle as a single node in a graph. These objects require periodic examination, but the problem is that they are scattered over a wide area, often over a country. Another problem is that the examinations cannot be done in an arbitrary period there is a minimum time between the examinations. The first part of the article describes the problem and shows some literature study. In the next part the mathematical model of the system is presented. The following chapter discusses the constraints of the optimization. The constraints are implemented with penalty functions as shown in the next chapter, which followed by the introduction of the general tabu search algorithm. Finally, the comparison of the previously developed evolutionary programming algorithm is presented. The researches presented in this article show the advantages of the developed evolutionary algorithm over the tabu search algorithm. In the future, application of other algorithms like the firefly algorithms is planned in this problem area.

KEYWORDS
Optimization, large scale networks, logistics maintenance, evolutionary programming, tabu search, multiple travelling salesman

1. INTRODUCTION
The significance of the technical inspection and maintenance systems are increasing in the field of globalized service industry. These systems ensure the safe and reliable operation of the production and service systems and they are important in the field of residential services like communal services, water, sewage, electricity, telecommunication services, monitoring and measuring devices, critical network control device or even elevator maintenance systems. The reliable, accident free and economical operation of these types of systems requires periodical inspections and maintenance requirements on site. The technical inspection tasks and maintenance in most cases require special knowledge and specially trained people. For example of the elevator inspection and maintenance systems where the technical inspection and maintenance are vital, and the proper operation can save lives; thus there are governmental regulations available [1].

The main problem is optimizing the fixed destination multiple depot multiple travelling salesman problem, with multiple tours and the special constraints emerging at technical inspection and maintenance systems. The pure multiple travelling salesman problem has a very extensive literature, but optimizing such a large systems with a lot of input parameters and boundary conditions has not had any. In the literature, one can found researches with multiphase optimizations [2] like clustering first and apply different multiple travelling salesman problem (MTSP) methods [3][4]. The clustering [5] and the partitioning [6][7] are widely used thanks to the speed of the algorithm, but these methods are less suitable to found the global optimum due to the nature of the multiphase optimizations.
The method we create can optimize large systems with one phase algorithm. Above all, it handles the special constraints of the technical maintenance systems we encountered in industrial projects. This also was an aim of the research: design optimization methods which take real world constraints into consideration. So using these optimization methods can result high savings in this area of logistics and we hope we can improve it with another methods in the future.

These constraints are the following:
- The locations of the objects are fixed.
- Experts have to return to their base location at the end of the cycle (mostly one day).
- Unlike the common MTSP (Figure 1), there are more than one operation have to be performed on a single object, so there are more routes (Figure 2).
- The experts have a limited, maximum capacity and also have a minimum capacity.

2. LITERATURE

The evolutionary programming [8] is a general problem solving algorithm like the genetic algorithm and a part of evolutionary algorithms family.

The most known algorithms of the evolutionary algorithms are the [9]: genetic algorithm; evolutionary programming; evolution strategy; genetic programming. All these algorithms have a common part: they handle a population. The population consists of individuals. One individual is one possible solution of the problem. The target is to get the best solution to the given problem. However, in most of the problems the algorithms did not have a chance the find the optimal solution, and one has to be satisfied with a quasi-optimal or a “good enough” solution.

The evolutionary programming is mainly used on heavily constrained problems like this problem. This method is also handling a population, but there are no limitations on the problem representation [10] like at the genetic algorithm where bit vectors describe the individuals [11]. Here, the problem described as the problem allows, or is it the best for computer algorithm. Moreover, the evolutionary programming is comparable in efficiency with the new algorithms like particle swarm optimization [12].

The developed algorithm solves the fixed destination multiple depot multiple route multiple travelling salesman problem, and optimize the number of salesmen in one phase. It can be used to solve large or very large problems. The explanation of the definition as follows; multiple salesmen: the experts; multiple depot: all the experts have different locations; multiple route: all the experts does more than one round routes; fixed destination: all the expert start and return to their initial location, and all the experts do the travel (generally) in one day cycles.

![Figure 1](image1.png)  
*Figure 1* Multiple experts and single inspection example (Legend: E: expert, O: object)

![Figure 2](image2.png)  
*Figure 2* Example with multiple routes with two inspections at the objects

The developed solution method based on a multi chromosome technique [13], which is not widely used in genetic algorithm, but it could be simply implemented in the evolutionary programming. In [14], four chromosomes used to represent the input and output fuzzy sets of a proportional-plus-derivative fuzzy logic controller. [15] used multi chromosome method to localize and quantify damage in truss structures.

The advantage of the multi chromosome technique that is reducing the size of the search space and the problem approach of this modelling technique is similar to the characteristic of the problem so it can be more problem specific, therefore more effective [16]).
3. MATHEMATICAL MODEL

The main input parameter of the optimization is the path matrix $L = [l_{ij}]$, the main output parameter is the assignment matrix: $Y = [y_{ij}]$, where $y = [1]$, the system elements are assigned together (1) or not (0).

The parameters of the objects: $p$: is the number of the objects; it is constant in this model, $L$ matrix defines the location of the objects, and the distance from the other system elements, $\kappa_{i}(i=1..p)$ is the mandatory inspection number per object.

The path travelled by the expert $i$ in a cycle $t$ can be described as:

$$l_i^t = l_{0,o_i^t(1)} + \sum_{c=1}^{[p_i^t]-1} (l_{o_i^t(c),o_i^t(c+1)})$$

and the total path travelled by the expert $i$ can be described as:

$$l_i^T = \sum_{t=1}^{T} l_{0,o_i^t(1)} + \sum_{c=1}^{[p_i^T]-1} (l_{o_i^T(c),o_i^T(c+1)})$$

where:

- $T$: is the number of cycles in the examined $\theta$ interval

The expenditures ($C$) of the experts ($S$) in a given period ($T$) can be described as:

$$C^S = \left[\sum_{t=1}^{T} \left( \sum_{j=1}^{s} l_i^t \right) c_u + \sum_{j=1}^{s} P_j \right] c_v$$

where:

- $c_u$: is the specific cost for one kilometer,
- $c_v$: the specific cost of an object.

Further in the article the specific cost is calculated with the multiplier 1, so only the distance is considered.

The target of the optimization is:

$$C^S \rightarrow min,$$

the expenditures have to be minimal.

3.1. Constraints

The number of the technical inspections and maintenance requirements could be prescribed by the maintenance plan, governmental regulations, or even the law in some cases where human life is endangered, like at the elevators [17]. The maintenance events cannot happen in an arbitrary period, there is a minimum period, which has to be defined for every object after when the next maintenance task could perform: $\tau^m = [\tau^m_i]_{i=1..p}$.

The interval of the inspections fulfils the constraint: $\tau^m_i * (\varepsilon_i - 1) \leq \theta$, where: $\varepsilon_i$ : is the number of the maintenance tasks of object $i$, $\theta$: is the examination period.

The performance ($P$) of the expert has to be between the defined minimum and maximum values:

$$P_{i, min} < P_i < P_{i, max}$$

The cycle time ($\tau_{max}$) generally one day - is also a constraint, in one cycle the expert visit the objects do the inspection and return to his base location:

$$\tau^c = \tau_{0,1} + \tau_1 + \sum_{t=2}^{t} (\tau^k_i + \tau_{i-1,i}) + \tau_{q,0}$$

where:

- $\tau^c$: is the interval when the expert start from his base location, visits the objects assigned to him and return; it is generally one day at the regional or countrywide maintenance systems:

$$\sum_{i=1}^{T} \tau_i^c \leq \theta$$

where:

- $\tau_{max}$: is the time interval of a cycle,
- $c^*$: is the number of objects has to visit in the cycle $t$,
- $\tau_{0,1}$: is the travel time to the first object from the starting location,
- $\tau_{q,0}$: is the travel time from the last object ($q$) to the base location of the expert,
- $\tau^k_i$: is the average inspection time of the object $i$.

The set of objects which have to inspect by the expert $c$, can be defined as:

$$O_c = \{o_i \mid y_{s,i} = 1; i = 1..p \}.$$
so the performance of the expert is:

\[ |O_c| = p_c , \] (9)

and the objects have to be inspected in one cycle (i), the subsets of the \( O_c \) :

\[ O^p_c \subseteq O^q_s , \] (10)

where:

\( O^p_s \) : is an ordered set, the objects assigned to the given expert, the ordering function is:

\[ a_p \in O_i, a_q \in O_i, a_p < a_q \text{ where } t_{op} < t_{oq}, \] (11)

where:

• \( t_{op} \) is the inspection time of \( a_p \),
• \( t_{oq} \) is the inspection time of \( a_q \),

so the set is ordered by the visiting time.

\[ |O^p_c | = c^p_c \leq p_c , \] (12)

\[ \bigcup_{t=1}^{T} O^p_c = O^c_s , \] (13)

and

\[ \bigcup_{s=1}^{p} O^q_s = O . \] (14)

However the expert performs more than one inspection on an object; so the object is counted in the sets defined at (8) as many times as the number of inspection has to be performed.

To determine the interval of the inspections the following distance function can be applied:

\[ d(o_i; o_j | o_i \in O^p_k; o_j \in O^q_l) = p - q , \] (15)

so based on the period constraint:

\[ \min \{d(o_i; o_j | o_i \in O^p_k; o_j \in O^q_l)\} \geq r^m_i . \] (16)

4. ALGORITHM

The developed evolutionary programming (already published in [18]) algorithm uses penalty functions to rate the goodness of the individuals [19]. The penalty function is one of the simplest and fastest way to rate the individual, so the goodness of the actual solution. In this algorithm, there are two different levels of penalty functions:

• local: the penalty function is applied to the expert,

• global: the penalty function is applied to the whole individual.

4.1. Penalty functions

There are three different local penalty functions:

• number of cycles penalty: when the expert do more route cycles than allowed (Figure 3),

\[ T_{max} \]

where:

\( P_{CC} \) : is the number of cycles penalty, \( T_{i} \) : is the cycles needed by the expert \( i \),

\[ T_{max} \] : is the maximum allowed cycles.

• few penalty : the expert has to get a minimal number of maintenance events (Figure 4),

\[ \min \{P_{i} \} \]

where:

\( P_{FP} \) : is the constant, penalty value of the 'few maintenance' violation,

\( P_{i} \) : is the actual performance of the expert \( i \), the number of the maintenance events performed by the expert,

\( P_{i, min} \) : is the minimal allowed inspection count of the expert \( i \).

• more penalty: the expert cannot get more maintenance events than his maximum capacity (Figure 5),

\[ B_{CC} = P_{CC} \cdot (T_i - T_{max}) \text{ if } T_i > T_{max} , \] (17)

\[ B_{FP} = P_{FP} \cdot (P_{i, min} - P_{i}) \text{ if } P_{i} > P_{i, min} \] (18)
which calculated after the local penalties:

There are three different global penalty functions, which calculated after the local penalties:

- near penalty: the maintenance events of one object cannot be arbitrarily close to each other (Figure 6).

\[ B_M = P_M \cdot \left( P_l - P_{l\text{ max}} \right) \text{ if } P_l > P_{l\text{ max}}, \] (19)

where:
- \( P_{M\text{P}} \): constant, the penalty value of the more maintenance violation
- \( P_l \): the actual performance of the expert \( i \), the number of the maintenance events,
- \( P_{l\text{ max}} \): is the maximum allowed inspection count of the expert \( i \).

There are three different global penalty functions, which calculated after the local penalties:

- near penalty: the maintenance events of one object cannot be arbitrarily close to each other (Figure 6).

\[ B_M = P_M \cdot \left( P_l - P_{l\text{ max}} \right) \text{ if } P_l > P_{l\text{ max}}, \] (19)

where:
- \( P_{M\text{P}} \): constant, the penalty value of the more maintenance violation
- \( P_l \): the actual performance of the expert \( i \), the number of the maintenance events,
- \( P_{l\text{ max}} \): is the maximum allowed inspection count of the expert \( i \).

The explanation of the different few and more penalty functions are the following:

- near penalty: The explanation of the different few and more penalty function. For example, the few penalty function is exponential, and the more penalty function is only linear, or sometimes a few penalty function gives is a death penalty value. The two constraints cannot be violated simultaneously so in the application it can be solved as one procedure.

- scatter penalty: This applied when maintenance events of an object are scattered among several experts (Figure 7).

\[ B_{SC} = \sum_{k=1}^{p} \left[ P_{SC} \cdot \text{count} \{ o_k \in O_x \wedge o_l \in O_y \} \right], \] (21)

where:
- \( P_{SC} \): is constant, the penalty value of scattered maintenance events,

If the scatter penalty switched off, the algorithm is not forced to assign every maintenance of one single object to one expert but is distributed between experts. As Figure 8 shows where object 3 and object 6 inspected by both experts.
of employed experts due to this penalty functions (Figure 9),

\[ B_S = \text{count}(P_i \neq 0|_{i=1,s}) \cdot P_{EC} \]  

where:
- \( P_{EC} \): is constant, the cost of one experts’ employment.

4.2. Fitness

So the calculation of the fitness of the individual is:

\[ F = C^S + B_S + B_{CC} + B_{SC} + B_F + B_M + B_N, \]  

where:
- \( F \): fitness of the individual,
- \( C^S \): is the cost of the route travelled by the experts,
- \( B_S \): is the cost of the employment of the experts,
- \( B_{CC} \): is the output of the number of cycles penalty function,
- \( B_{SC} \): is the output of the scatter penalty function,
- \( B_F \): is the output of the few penalty function,
- \( B_M \): is the output of the more penalty function,
- \( B_N \): is the output of the near penalty function.

The target of the optimization application:

\[ F \rightarrow \text{min}. \]  

4.3. Operators

The used operators can also be divided to two groups due to the multi chromosome characteristic of the algorithm:
- inner expert operators which are local operators and,
- cross expert operators which are global operators.

There are three types of inner expert mutation operators:
- gene swap: two random genes are swapped (Figure 10)
- gene sequence reversion: the gene sequence is reversed between two random indexes (Figure 11)
- gene insertion operator: a randomly chosen gene inserted into a randomly chosen position (Figure 12)
- cross expert gene swap: which swaps two genes between experts. The second expert and the position of the other gene are chosen randomly (Figure 13),
- cross expert gene sequence change: the algorithm swapping a randomly chosen but continuous gene sequence with a randomly chosen expert also randomly chosen gene sequence (Figure 14),
• cross expert gene contraction: where a random amount of genes inserted to a randomly chosen expert from the end of the chromosome (Figure 15).

![Figure 15 Cross expert sequence swap](image)

4.4. Survivor selection

The survivor selection is the process responsible for choosing the members of the descendant population. The survivor selection we used uses elitism; the best individual (elitist) always survives [20]. First, the algorithm searches for the best individual and copy it to the descendant population. It is called elitism; the best individual (elitist) always survives. The evolutionary programming typically uses stochastic tournament survivor selection, and there are some new methods like the planned tournament selection [21], but these methods need some further researches for speed and efficiency.

The simplest of the tournament selection - and therefore the fastest - is when randomly choose two individuals (1+1 strategy), one from the original and one from the mutated population and the fittest one wins, and it will be copied into the next generation. This process is repeated until the next population is.

Avoiding the local optima all the genetic methods try to maintain the diversity among the individuals. In this algorithm, one can parameterize how many individual in the descendant population will be filled with random generated individuals.

The tournament process:

• randomly choose an individual from the original and another from the mutated population,
• the individual with the best fitness value is inserted into the descendant population,
• repeat the process until the population is filled up to the required count.

The selection process fills the descendant population up to a parameterized value. The rest are filled with random generated individuals, which helps to avoid local optimum (Figure 16).

Avoiding the local optima all the genetic methods try to maintain the diversity among the individuals. In this algorithm, one can parameterize how many individuals in the descendant population will be filled with random generated individuals.

5. TABU SEARCH

The tabu search is a widely used, problem independent metaheuristic search method, mainly in combinatorial optimization [22]. The tabu search using a ‘one element’ population, the search operator like at the genetic algorithm is the mutation, but in this case there are no inheritance and crossover methods. The tabu search method is using a list, which contains some of the last examined solutions.

Some variants of the tabu search use more tabu lists to simulate short term and long term memory [23]. The only parameter is the length of the tabu list.

First the algorithm has to check if the new solution, which created by mutation operators, is on the tabu list. If it is on the tabu list then it is discarded, and if it is better than the last best solution it will be accepted as the new best solution and the old one goes to the tabu list. If the tabu list reached beyond its maximum capacity, the oldest elements of the tabu list would be cleared [24] (Figure 17).
In the tabu search, the same operators and penalty functions have been used like at the evolutionary programming method.

The tabu search algorithm was tested on several functions have been used like at the evolutionary programming method. The length of the tabu list was set to 500, at the genetic method the population size was 500 by 50 random generated individually and at the last case the population size was 10 by 1 random generated individually to reach the same speed like at the first three cases.

The first test problem is a small size problem with 2 experts and 50 objects and 1 examination per object.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Target func.</th>
<th>Relative diff.</th>
<th>Solving time</th>
<th>Penalty remaining</th>
<th>Iteration count</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP</td>
<td>4008,85</td>
<td>100,00%</td>
<td>0:13:06</td>
<td>0</td>
<td>9100</td>
</tr>
<tr>
<td>T</td>
<td>5491,27</td>
<td>73,00%</td>
<td>0:13:22</td>
<td>0</td>
<td>15000</td>
</tr>
</tbody>
</table>

Table 1 Comparison of the tabu search and evolutionary programming (Legend: EP: Evolutionary programming, T: Tabu search)

The second test problem is also a small sized problem with 50 objects and 1 examination per object.

<table>
<thead>
<tr>
<th>Alg.</th>
<th>Target func.</th>
<th>Relative diff.</th>
<th>Solving time</th>
<th>Penalty remaining</th>
<th>Iteration count</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP</td>
<td>4483,96</td>
<td>100,00%</td>
<td>0:24:35</td>
<td>0</td>
<td>16434</td>
</tr>
<tr>
<td>T</td>
<td>159581,79</td>
<td>2,81%</td>
<td>0:28:57</td>
<td>3</td>
<td>30000</td>
</tr>
</tbody>
</table>

Table 2 Results of the tabu search and evolutionary programming comparison.

The results show (Table 2) that the tabu search gives worse result compared to the evolutionary programming algorithm in this case also, but in this case, the tabu search algorithm was run up to 30,000 iterations and almost 4 minutes more than the evolutionary programming algorithm. In this case, the tabu search solution is much worse than the evolutionary programming method.

The next example is a more complex example with 3 experts and more than one examination per object. The examination count is defined randomly from 2 to 4. The evolutionary programming method was run up to 20000 iterations the tabu search was run up to 65000 iterations, but the running time was the same.

<table>
<thead>
<tr>
<th>Alg.</th>
<th>Target func.</th>
<th>Relative diff.</th>
<th>Solving time</th>
<th>Penalty remaining</th>
<th>Iteration count</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP</td>
<td>217534,75</td>
<td>100,00%</td>
<td>9:01:12</td>
<td>11</td>
<td>200000</td>
</tr>
<tr>
<td>T</td>
<td>495470,75</td>
<td>43,90%</td>
<td>9:02:10</td>
<td>14</td>
<td>650000</td>
</tr>
</tbody>
</table>

Table 3. Results of the tabu search and evolutionary programming comparison.

The results (Table 3) (Fig. 18, Fig. 19) show that the tabu search algorithm gives worse results that the evolutionary programming method in this case also at the same running time. The tabu search approximates the result to 43.9 percents.

The next example is a complex large problem with 1000 objects, three experts and 2-4 examinations per object. Here, the evolutionary programming method
was run for 5500 iterations, the tabu search was run for 29000 iterations but same time as the evolutionary programming method.

<table>
<thead>
<tr>
<th>Alg.</th>
<th>Target func.</th>
<th>Relative diff</th>
<th>Solving time</th>
<th>Pen. rem.</th>
<th>Iter. count</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP</td>
<td>138245712.9</td>
<td>100.00%</td>
<td>0:57:05</td>
<td>631</td>
<td>5500</td>
</tr>
<tr>
<td>T</td>
<td>138781405.3</td>
<td>99.61%</td>
<td>0:57:46</td>
<td>645</td>
<td>29000</td>
</tr>
</tbody>
</table>

Table 4 Results of the tabu search and evolutionary programming comparison.

The results (Table 4) show that the tabu search algorithm gives the worse result here as the evolutionary programming method as well, but here the advantage of the evolutionary programming algorithm was not big as in the other examples. The tabu search approximated the target function up to 99.61 percents. In this case, the goodness of the solution cannot be decided based on the figures because of the complexity of the problem. Both algorithms were tested on the same computer with Intel Core i7-870 2.93 GHz CPU and the optimization program was implemented in C# programming language.

7. CONCLUSION

The tabu search gives worse results than the evolutionary programming algorithm developed in this research for all cases. Although, at the last example, it approximated the results of the evolutionary programming method closely, but at the second example it gives only the small percent of the results of the evolutionary programming method.

The advantage of the tabu search that it does not calculate the fitness of the already examined elements – which found on the tabu list – but at this problem it has more disadvantages because the numbers of the neighbouring elements created by the operators are huge. Till at searching general function the neighbourhood elements can be easily identified, which elements are the ones which can be accessed by one step in the search space. Here, the number of the neighbourhood elements is very high, so the determination of these elements is virtually impossible, and the neighboring element number is highly increasing with the problem size. The reason is that the set of the neighboring elements is what the currently used six operators calculate with all parameter combinations. Because of this there is a minimal hit on the tabu list. In practice during an examination of a medium size problem there was not any hit on the tabu list, but the searching in the tabu list, the comparison of the complex elements, putting an element in the tabu list, memory allocation, element copy, takes a great amount of time which is comparable to the calculation of the fitness of one element.

8. FURTHER RESEARCH

The optimization method has great potentials toward the further improvements. First some improvements in the method himself:

- try new mutation methods,
- try new tournament selection methods.
- Second some improvements which have a high impact on speed:
- massive parallelization in a computing cloud,
- examine the implementation possibilities on fast graphics processing units (GPU) or a GPU cluster.

Try to solve the problem with other optimization methods if possible, like the firefly optimization [25] from the swarm optimization family and compare it with the developed evolutionary algorithm.

ACKNOWLEDGMENTS

The research was supported by the TÁMOP 4.2.4.A/2-11-1-2012-0001 priority project entitled “National Excellence Program - Development and operation of domestic personnel support system for students and researchers, implemented within the framework of a convergence program, supported by the European Union, co-financed by the European Social Fund. The research was supported also by the Hungarian Scientific Research Fund OTKA T 75678 and T 109860 projects and was partially carried out in the framework of the Center of Excellence of Innovative Engineering Design and Technologies at the University of Miskolc.

REFERENCES


