PERTURBATION ALGORITHM
FOR MINMAX REGRET MINIMUM SPANNING TREE PROBLEM

Mariusz Makuchowski
Institute of Computer Engineering, Control and Robotics, Wrocław University of Technology, Wybrzeże Wyspiańskiego 27, 50-370, Wrocław, Poland

This paper analyses the problem of finding a robust spanning tree. The problem consists in determining a minimum spanning tree of a graph with uncertain edge costs. We should determine such a spanning tree that minimizes the cost difference between a selected tree and the optimal tree. While doing this, all possible realizations of edge costs should be taken into account. This issue belongs to the class of NP-hard problems. In this paper an algorithm based on the cost perturbation method dedicated to the analysed problem has been proposed. The paper also contains results of numerical experiments testing the effectiveness of the proposed algorithm and compares it with algorithms known in the literature. The research is done on a large number of various test examples taken from the literature.

1. Introduction

In the real world we often have to make a decision when the parameters are unknown, e.g. a decision about purchasing a summer swimming costume without a guarantee of a warm summer. The quality of the decision made can be evaluated only with hindsight. What’s more, it’s only from such a perspective that we see what the optimal action would have been on our part. Because at the time of making a decision we are unable to evaluate its actual value, often indirect evaluations are used, such as the worst scenario analysis or expected value analysis. In this paper the criterion evaluating a given decision is maximal regret analysis. What we call regret is the difference between the value obtained from a decision made and the hypothetical value obtained for the decision which with hindsight would have been regarded as optimal. As we are unable to foresee the future, we try to make sure that the biggest regret that may happen to us is possibly the smallest.

Some easy problems, i.e. belonging to the P class, with precise data, when the parameters are changed into imprecise ones and the criterion is transformed to regret of this criterion, become issues that belong to the class of NP-hard problems. This is also the case of the NP-hard minmax regret minimum spanning tree problem examined in this paper, which derives from a classical easy minimum spanning tree problem.

2. Robust tree problem

One of the basic problems of operations research is determining a minimum cost spanning tree. A spanning tree \( t \) of a connected, undirected graph \( G = (V, E) \) is any acyclic connected subgraph of this graph. We will use \( \Gamma \) to denote the set of all
the spanning trees of a given graph. The number of the spanning trees of a graph depends both on its size and structure. For complete graphs, it depends exponentially on the number of nodes in the graph and is:

$$\text{det}(\Gamma) = |\mathcal{V}|^{\frac{|\mathcal{V}|}{2} - 2}. \quad (1)$$

For graphs with weighted edges, we can determine the cost of any spanning tree as the sum of the weights of its edges. The problem of minimum cost spanning tree consists in choosing from all possible trees spanning a given graph the tree with the lowest cost. This problem is solved in polynomial time by the algorithm proposed in Prim (1957) or Kruskal (1956).

In the analysed problem weighs of the edges are not known. For each edge $e \in \mathcal{E}$ the range of its possible realizations is given $[c^-_e, c^+_e]$. A specific realization of all the edge weights is called a scenario and denoted by $s$. We will use $\Omega$ to denote the set of all possible scenarios. The cost of edge weigh $e \in \mathcal{E}$ in the scenario $s \in \Omega$ will be denoted by $c^s_e$; $c^s_e \in [c^-_e, c^+_e]$. For a given tree $\Gamma \in \Gamma$ its cost depends on the scenario $s \in \Omega$ and is:

$$F(t, s) = \sum_{e \in \mathcal{E}} c^s_e. \quad (2)$$

For the scenario $s \in \Omega$ we can determine a minimum cost spanning tree with the total cost of:

$$F^*(s) = \min_{\Gamma \in \Gamma} F(t, s). \quad (3)$$

For each tree $t \in \Gamma$ and scenario $s \in \Omega$ we can determine regret, which is the difference between the cost of a given tree and the cost of the minimum tree for a given scenario:

$$Z(t, s) = F(t, s) - F^*(s). \quad (4)$$

The maximal regret of the tree $t \in \Gamma$ is defined in the following way:

$$Z(t) = \max_{s \in \Omega} Z(t, s) = \max_{s \in \Omega} \{F(t, s) - F^*(s)\}. \quad (5)$$

The problem analysed in this paper is to find the tree $t^* \in \Gamma$ with the minimum value of maximal regret:

$$t^* \in \arg\min_{t \in \Gamma} Z(t). \quad (6)$$
3. Overview of the literature

Aron I., Hentenryck (2003) present an evidence that the problem analysed is NP-complete. In Bezrukov at al. (1996) a special case is analysed in which all the edge costs can have values from the interval [0,1]. The problem so reduced is called the central spinning tree problem and also belongs to the class of NP-hard problems.

The first method for providing an exact solution to the problem examined was proposed in Yaman at al. (2001). It was based on a mixed integer programming (MIP) and using it the authors solved instances having up to 25 nodes in a graph. In Aron and Hentenryck. (2002) and Montemanni and Gambardella (2005) algorithms based on the branch and bound method were proposed which were able to provide solutions for graphs having up to 20-40 nodes.

In Kasperski and Zieliński (2006) a 2-approximation, polynomial time, construction algorithm was proposed, whereas Nikulin (2007) presented an approximate improvement algorithm based on the simulated annealing method. The most efficient in terms of the quality of solutions for large instances (which cannot be solved with exact algorithms) is the algorithm proposed in Kasperski et al. (in print) based on the taboo search method.

4. Perturbation algorithm

Perturbation algorithms are a modification of existing algorithms, most often construction algorithms. The idea of a perturbation algorithm consists in multiple perturbation of input data and then solving the received problem with the help of a certain algorithm, called later a base algorithm. The solution of the disturbed problem obtained in this way is tested on the original, unchanged input data. Because perturbations are random, a perturbation algorithm can perform a whole series of identical iterations. After performing a fixed number of iterations, the algorithm stops and returns the best found solutions (the best in terms of values of the target function for the original data). The pseudo code of the perturbation algorithm is shown in figure 1.

| Step 1: | enter the problem data D |
| Step 2: | repeat n-times steps 2a,2b,2c |
| Step 2a: | randomly perturbate the original data D |
| | creating data D' |
| Step 2b: | initiate Base Algorithm on perturbed data |
| | D' obtaining solution x' |
| Step 2c: | if x' for original data D is a better |
| | solution than the best found solution x* |
| | then replace x*=x' |
| Step 3: | return the best found solution x* |

Fig. 1. Pseudo code of a perturbation algorithm

Source: own work
Undoubtedly, the basic element of each perturbation algorithm is a fast construction algorithm which provides high quality solutions. In the problem analysed, these will be two algorithms, AM and AU, described in Kasperski et al. (2005). For discrete optimization problems, in which the input data (just like in the problem examined) is a graph with weighted edges, various variants of cost perturbations are applied. Ribeiro at al. (2001) propose algorithms perturbing all the weights of the edges. Moreover, appropriate perturbation results in intensification or diversification of calculations. Intensification of calculations is obtained by perturbing the costs in such a way that they promote the edges chosen in the previous iterations of the algorithm, whereas diversification is obtained by perturbing the costs in the opposite way.

The first perturbation algorithm presented in the paper is called PMU. It perturbs the upper bounds of the intervals of the weights of the values $[c_e^-, c_e^+]$ such a way that it selects randomly $c_e^+$ a new value of the upper bound from the interval:

$$c_e^+ \in [c_e^+, (1+i/maxIter)\cdot c_e^+]$$

where $i$ is the number of the current iteration and $maxIter$ is the number of the last iteration of the algorithm. During the research the ways of perturbation described in Ribeiro at al. (2002) have also been checked, but they haven’t given significantly better results compared to the presented version. As the base algorithm two construction algorithms AM and AU have been used simultaneously. Simultaneous use of these two algorithms will be called further in the paper the AMU algorithm. The AM algorithm provides a minimum spanning tree for a graph in which the values of the weights of the edges are exactly $c_e = (c_e^--c_e^+)/2$, whereas the AU algorithm provides a minimum spanning tree for a graph with the weights of the edges which are $c_e = c_e^+$. During the perturbation algorithm running each of the solutions obtained with the help of the AM and AU algorithms is tested on the original data and may become the best found, and thus finally returned solution.

A natural application of perturbation algorithms is using them in all kinds of improvement algorithms or local search algorithms. Such an algorithm is shown in fig 2.
In this case, a perturbation algorithm serves as a generator of starting solutions for a proper improvement algorithm. By selecting the parameters of a data perturbation, we can regulate the distribution of starting points in the whole space of permissible solutions. Greater distribution in the space can be obtained by increasing the perturbation. We should however bear in mind that increased perturbation results in decreased quality of generated solutions.

The second of the presented algorithms, called PMU+LS, is a PMU perturbation algorithm combined with a local search algorithm. It generates a perturbation solution obtained with the help of the PMU algorithm proposed above, and then enhances it with an improvement algorithm until an optimally local solution is found. In the PMU+LS algorithm, the number of PMU algorithm perturbations is limited to one and consists in changing the upper bounds of the intervals of the weights of the values $[c_r^-, c_r^+]$ so that it selects randomly a new value of the upper bound from the interval: $\tilde{c}_r^+ \in [c_r^-, 2c_r^+]$.

The neighborhood of the current solution (the spanning tree of a given graph $G$) applied in the used improvement algorithm is a set of spanning trees of a graph $G$ which differ in exactly one edge from the current solution, that is:

$$N(t) = \{t' \in \Gamma : |E(t) \setminus E(t')| = 1 \} \mathrm{dla} \ t \in \Gamma.$$  \hspace{1cm} (8)

The stopping criterion in the algorithm PMU+LS is a number of performed iterations. This algorithm will be compared to its classic equivalent called AR+LS. We will use AR to denote an algorithm providing a random spanning tree. We also examined two improvement algorithms based on the taboo search method: AR+TS, PMU+TS. AR+TS is a classic taboo search algorithm with a multiple start from random solutions. The third of the proposed algorithms is PMU+TS, in which one course algorithm PMU has been used to generate starting solutions. The implemented TS algorithm has been described in detail in Kasperski et al. (in print).
5. Numerical research

All the presented research results, especially running times of algorithms, have been obtained on a PC computer equipped with the processor Core2Duo E6750 2.66GHz, 4GB of RAM, running under Windows 7. The algorithms have been programmed in the environment DEV C++ 4.9.9.1.

The presented algorithms have been compared using examples generated in accordance with a conception developed by researchers dealing with the problem analyzed. The first 2 classes of examples Yaman1, Yaman4 were proposed in Yaman et al. (2001). This work describes the construction of six classes of examples, of which class 1 and class 4 have been chosen for the purpose of the research described this paper. Next two classes of examples Hente1, Hente2 were described in Aron and Hentenryck (2002). Next two classes Monte1, Monte3 were generated according to the formula in Montemanni, Gambardella (2004), but out of 3 classes of proposed instances class 1 and class 3 have been chosen. Each of the classes consists of 10 groups and each group numbers 10 instances. Graphs within a given group have the same number of nodes, which varies from 10 to 100. A total of 600 test examples have been used in the research.

In the literature, Kasperski (2008), Kasperski et al. (in print), there are also other, very difficult test examples (Ka, La) of the special case, which is instances of the central tree problem. They are however deliberately omitted due to the fact that in these instances all the weights in graphs are described in the same interval [0,1]. The difficulty for these instances results only from their topology of graph, not, as in previous instances, from different intervals of edge weights. Because the perturbation algorithm PMU presented in the paper is based on AMU algorithms, which are influenced by the intervals of the weights – both of the algorithms mentioned are completely blind when solving these examples. Thus AMU and PMU algorithms provide random solutions for these instances and do not differ from an AR algorithm. So there is no point in comparing the algorithms AMU, PMU, AR or their extensions using the above-mentioned test examples.

Below we will compare an average relative deviations of a given group of test instances of the examined algorithms. For this purpose, for each of the instances a reference solution, obtained after many hours of calculations, will be determined. For smaller instances, exact algorithms have been used based on the division and cuts method, while for the other instances – algorithms based on the taboo search method initiated with various controlling parameters. The solution of an instance found in this way has been denoted by \( t_{REF} \). Then, the examined algorithm \( A \) has been run and the solution produced by it has been denoted by \( t^A \). The value of a criterion function of the solutions so obtained has been calculated according to the formula (5) and denoted by \( Z(t_{REF}) \) and \( Z(t^A) \) respectively. Next, based on these values the deviation from the reference solutions of the algorithm \( A \) (for one instance) has been calculated as:

\[
\rho^A = \frac{Z(t^A) - Z(t_{REF})}{Z(t_{REF})} \cdot 100\%.
\]
The paper presents \( \bar{\rho} \) average values of deviations (8) obtained by the examined algorithms within a given test class (100 instances – 10 groups with 10 instances of the same size in each).

**Table 1. Average deviation from the reference solutions \( \bar{\rho} [%] \) for the algorithms AMU, AR, PMU, AR(300s) and PMU(300s)**

<table>
<thead>
<tr>
<th>Class of instances</th>
<th>AMU t=30s</th>
<th>AR t=30s</th>
<th>PMU t=30s</th>
<th>AR (300s)</th>
<th>PMU (300s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yaman1</td>
<td>0.80</td>
<td>223.23</td>
<td>7.05</td>
<td>147.99</td>
<td>0.07</td>
</tr>
<tr>
<td>Yaman4</td>
<td>0.23</td>
<td>316.60</td>
<td>5.90</td>
<td>218.53</td>
<td>0.02</td>
</tr>
<tr>
<td>Hente1</td>
<td>3.19</td>
<td>1302.30</td>
<td>50.85</td>
<td>641.04</td>
<td>1.69</td>
</tr>
<tr>
<td>Hente2</td>
<td>3.24</td>
<td>631.28</td>
<td>42.84</td>
<td>165.86</td>
<td>1.25</td>
</tr>
<tr>
<td>Monte1</td>
<td>0.72</td>
<td>191.56</td>
<td>70.57</td>
<td>15.74</td>
<td>0.07</td>
</tr>
<tr>
<td>Monte3</td>
<td>1.75</td>
<td>462.83</td>
<td>10.31</td>
<td>233.91</td>
<td>0.43</td>
</tr>
<tr>
<td>Total</td>
<td>1.66</td>
<td>521.30</td>
<td>31.25</td>
<td>237.18</td>
<td>0.59</td>
</tr>
</tbody>
</table>

Source: own work

In the first test, the construction algorithms AMU, RAND, PMU, RAND(300s), PMU(300s) have been compared. The algorithm AR(300s) means 6100 runs of the AR algorithm and choosing the best provided solution. The algorithm denoted by PMU is a perturbation algorithm with a single data turbulence and a single run on perturbed data of AM and AU algorithms. In the algorithm PMU(300s) there are 895 calls of the PMU algorithm. In both the perturbation algorithms data perturbation is performed in accordance with the formula (7).

The number of iterations in the RAND(300s) and PMU(300s) algorithms has been chosen experimentally in such a way that the time of generating and calculating all the 600 test instances is in both cases 300s (270s of the running of the proper algorithm and 30s for the input-output operations). The remaining algorithms had an extremely short running time and with the input-output operations the total running time for all instances has been around 30s. The obtained results of the research of the first test have been shown in table 1.

In the second test two local search algorithms (AR+LS)(300s) and (PMU+LS)(300s) and two algorithms based on the taboo search method (AR+TS)(300s) and (PMU+TS)(300s) have been compared. The first two of the examined algorithms, after finding the locally optimal solution, continue further search out of the solutions provided with the help of AR and PMU algorithms respectively, while the next two search the space around the found local minimum in accordance with the taboo search method. Only after 20 iterations without improvement they analogically perform a restart from the solutions generated by AR and PMU. The number of iterations in all the examined algorithms is selected (at
around 60) in such a way that the running time of each of them on all the 600 instances is 300s.

Table 2. Average deviation from the reference solutions $\bar{\rho} [%]$ for the algorithms (AR+LS)(300s), (PMU+LS)(300s), (AR+TS)(300s), (PMU+TS)(300s)

<table>
<thead>
<tr>
<th>Class of instances</th>
<th>(AR+LS)(300s)</th>
<th>(PMU+LS)(300s)</th>
<th>(AR+TS)(300s)</th>
<th>(PMU+TS)(300s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\bar{\rho} [%]$</td>
<td>$\bar{\rho} [%]$</td>
<td>$\bar{\rho} [%]$</td>
<td>$\bar{\rho} [%]$</td>
</tr>
<tr>
<td>Yaman1</td>
<td>6.96</td>
<td>0.00</td>
<td>4.73</td>
<td>0.00</td>
</tr>
<tr>
<td>Yaman4</td>
<td>12.52</td>
<td>0.00</td>
<td>8.65</td>
<td>0.00</td>
</tr>
<tr>
<td>Hente1</td>
<td>0.47</td>
<td>0.00</td>
<td>0.17</td>
<td>0.01</td>
</tr>
<tr>
<td>Hente2</td>
<td>0.12</td>
<td>0.00</td>
<td>0.10</td>
<td>0.01</td>
</tr>
<tr>
<td>Monte1</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Monte3</td>
<td>1.89</td>
<td>0.00</td>
<td>0.88</td>
<td>0.00</td>
</tr>
<tr>
<td>Total</td>
<td>3.66</td>
<td>0.00</td>
<td>2.42</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Source: own work

The results contained in table 1 show that a single random perturbation of input data makes a dedicated algorithm worsen its effectiveness from $\bar{\rho}_{AMU} = 1.66[\%]$ < $\bar{\rho}_{PMU} = 31.25[\%]$. Although the modified operation of the PMU algorithm has random elements, the quality of the obtained solution is significantly better than the completely random solutions $\bar{\rho}_{PMU} = 31.25[\%] < \bar{\rho}_{AR} = 521.30[\%]$. In the proposed PMU algorithm, multiple running improves the provided solution, which is not the case with the original AMU algorithm. Therefore the proper test consisted in a sequential running of the PMU algorithm for the period of 300s. The PMU(300s) algorithm provided significantly better solutions than the original algorithm AMU; $\bar{\rho}_{PMU(300s)} = 0.59[\%] < \bar{\rho}_{AMU} = 1.66[\%]$. A similar method for the random AR algorithm has not been bringing good results $\bar{\rho}_{AR(300s)} = 237.18[\%]$.

The results of the second test (table 2) show that the use of the proposed PMU algorithm instead of the random AR algorithm in improvement algorithms significantly improves their effectiveness. For local search algorithm starting from random solutions within 300s, solutions with the average deviation of $\bar{\rho}_{LS+AR(300s)} = 3.66[\%]$ have been obtained, and replacing the AR method with the PMU method in this algorithm resulted in the algorithm providing solutions with zero deviation $\bar{\rho}_{LS+PMU(300s)} = 0.00[\%]$. Similarly, when the AR method has been replaced by the PMU method in the TS algorithm, the deviation of the solutions
provided within 300s decreased to zero; $p^{(TS+AR)(300s)} = 2.42\%$, $p^{(TS+PMU)(300s)} = 0.00\%$.

The advantage of the PMU + LS algorithm, over the algorithm TS + AR is only apparent. In the tested examples, you can easily determine the trajectory of reaching the reference solution. PMU + LS algorithm starts with a solution closer to the reference solution than the algorithm AR + TS. The distance is understood here as the minimum number of moves needed to move from one solution to another. Since reaching the reference solution is not a problem for the presented algorithm, an algorithm starting from the solutions, which are closer to the reference solution, quickly reaches the target. The prevalence of the algorithm TS is clearly visible (see table 3) in difficult examples, in which it is hard to find a way to determine a reference.

Table 3. Average deviation from the reference solutions $\overline{\rho} (%)$ for the algorithms 
(AR+LS)(300s), (PMU+LS)(300s), (AR+TS)(300s), (PMU+TS)(300s)  

<table>
<thead>
<tr>
<th>Class of instances</th>
<th>(AR+LS)(300s)</th>
<th>(PMU+LS)(300s)</th>
<th>(AR+TS)(300s)</th>
<th>(PMU+TS)(300s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\overline{\rho} (%)$</td>
<td>$\overline{\rho} (%)$</td>
<td>$\overline{\rho} (%)$</td>
<td>$\overline{\rho} (%)$</td>
<td></td>
</tr>
<tr>
<td>Ka</td>
<td>14.00</td>
<td>14.55</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>La</td>
<td>14.05</td>
<td>14.32</td>
<td>1.55</td>
<td>0.92</td>
</tr>
<tr>
<td>Total</td>
<td>14.02</td>
<td>14.44</td>
<td>0.77</td>
<td>0.46</td>
</tr>
</tbody>
</table>

Source: own work

6. Summary

For the analysed minmax regret minimum spanning tree problem, a perturbation algorithm PMU has been proposed and based on it a modification of improvement algorithms.

An advantage of the proposed perturbation algorithm PMU over the classic AMU approach is the possibility of improving the average quality of provided solutions at the cost of the time of the algorithm running. For the algorithm running for exactly the same time as the AMU algorithm, the provided solutions are of worse quality. If we increase the running time of the PMU algorithm several times, we can obtain solutions whose quality cannot be received using the classic AMU solution.

Another serious application of the presented PMU algorithm is using it in improvement algorithms, particularly in advanced improvement algorithms extended by the mechanisms of diversification based on multiple starts. After replacing the AR method, which provides random start solutions, with the proposed PMU improvement algorithm, improvement algorithms significantly increase their effectiveness, that is in the same time they provide better, in terms of the value of optimized criterion, solutions.
This work was supported by Polish Committee for Scientific Research, grant N N206 492938.

References


