Paper

Monte Carlo Tree Search Algorithm for the Euclidean Steiner Tree Problem

Michał Bereta

Institute of Computer Science, Cracow University of Technology, Cracow, Poland

https://doi.org/10.26636/jtit.2017.122017

Abstract—This study is concerned with a novel Monte Carlo Tree Search algorithm for the problem of minimal Euclidean Steiner tree on a plane. Given *p* points (terminals) on a plane, the goal is to find a connection between all the points, so that the total sum of the lengths of edges is as low as possible, while an addition of extra points (Steiner points) is allowed. Finding the minimum Steiner tree is known to be np-hard. While exact algorithms exist for this problem in 2D, their efficiency decreases when the number of terminals grows. A novel algorithm based on Upper Confidence Bound for Trees is proposed. It is adapted to the specific characteristics of Steiner trees. A simple heuristic for fast generation of feasible solutions based on Fermat points is proposed together with a correction procedure. By combing Monte Carlo Tree Search and the proposed heuristics, the proposed algorithm is shown to work better than both the greedy heuristic and pure Monte Carlo simulations. Results of numerical experiments for randomly generated and benchmark library problems (from OR-Lib) are presented and discussed.

Keywords—Euclidean Steiner tree problem, MCTS, Monte Carlo Tree Search, UCT algorithm.

1. Introduction

One of the most recent achievements in the field of computational intelligence was the victory of the AlphaGo program in the game of Go against human champion Lee Sedol. The game of Go had been a challenge for computer scientists for many years. AlphaGo has been based on the mixture of deep learning algorithms [1] and the Monte Carlo tree search approach [2]. Monte Carlo tree search (MCTS) has proved to be a promising new search technique. Although the most spectacular successes of MCTS come from the field of playing agents, MCTS is a general search technique, and attempts have already been made to apply this technique to other problems, such as planning or optimization.

In this work, MCTS is applied to the problem of finding the minimal Steiner tree on the Euclidean plane (ESTP). ESTP is formulated as follows. Given p points called terminals, connect them so that the sum of lengths of edges is minimal. Additional intermediate points called Steiner points can be added. If adding Steiner points is not allowed, the problem becomes the task of finding the minimum spanning tree (MST) for the given set of terminals, which can be solved efficiently by algorithms such as Prim [4]. However, ESTP is np-hard [5]. Figure 1 shows a minimal spanning tree and a Steiner tree for a sample set of points. The sum of lengths of all edges is smaller for the Steiner tree than for the minimal spanning tree.

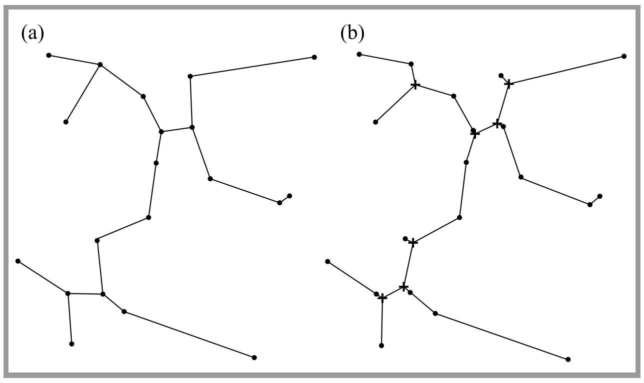

Fig. 1. A minimal spanning tree (a) and a Steiner tree (b) for a set of points; Steiner points are marked with "+".

Many practical problems require to solve a two-dimensional ESTP on a plane, or other versions of Steiner tree. Therefore, the issue evokes broad interest. Three-dimensional problems are also of practical concern. Such problems emerge in VLSI chip layout design, distribution network planning [6], wireless and sensor networks [7], robotics and molecular structure modeling [8]. A good review of Steiner tree applications can be found in [7], robotics and molecular structures modeling [8]. Another good review of Steiner trees applications can be found in [5]. Although exact algorithms exist that enable to solve ESTP on a plane, such as those proposed by Melzak [9], [10], Trietsch and Hwang [11] and Smith [12], they are time-consuming, and there is a need for new algorithms. New heuristics are still being proposed [13]. Evolutionary algorithms have also been proposed for various types of Steiner problems, such as [14]. The MCTS algorithm proposed for ESTP is, to the best of the author's knowledge, truly unique.

Motivations and contributions of this work are as follows. Firstly, it is the first time the Upper Confidence bound for Trees (UCT) algorithm has been applied to the Steiner problem. Secondly, while exact algorithms related to the Euclidean Steiner problem in 2D exist, the proposed algorithm can serve as a heuristic to provide fast preliminary solutions.

Many Steiner problems, such as in higher dimensions, remain unsolved, and the proposed algorithm can be treated as the first attempt to approach these problems with MCTS.

The rest of the paper is organized in the following manner. In Section 2, MCTS is introduced as a general search algorithm together with its more specific version, which uses upper confidence bound (UCB) as a policy to guide the search process. In Section 3, the most important properties of Euclidean Steiner trees are described, which enable to formulate a simple heuristic and a greedy search technique. This heuristic is further combined with MCTS in Section 4 to propose a new algorithm for ESTP. Several possible improvements are also discussed. In Section 5, the results of numerical experiments are presented and analyzed. By using proper statistical procedures, the results of the proposed MCTS for ESTP are compared with the results of the greedy search method and simple Monte Carlo simulation. Additionally, the ability of the proposed algorithm to find the correct solutions is tested on the benchmark problems from OR-Lib [15], for which the exact solutions are known. Finally, the conclusions are drawn in Section 6.

2. Monte Carlo Tree Search

In many instances, searching for a solution to a given problem can be represented as traversing the nodes of a tree. In most cases it is impossible to visit all the nodes efficiently. Therefore, heuristics are used to make a decision about the most promising nodes that should be visited to maximize the reward (e.g. the probability of winning the game). In Monte Carlo (MC) simulations the approach is based on randomly sampling the search space and averaging the results. It has proved to be successful in many problems. In the game theory, it was proved that the process of MC simulation could accurately estimate the expected reward of an action [16]. The idea of combining Monte Carlo simulation with searching the tree was proposed for the first time in 2006 by Coulom [17]. The core idea of the proposed method is to take random samples in the search space of the given problem domain, and to build the search tree based on the results. The search tree itself is involved in the simulation, and its structure is constantly expanded by deepening those parts that seem to be more promising, based on MC estimates available at a given time. The whole process of MC tree search is usually described as consisting of four phases, which are described below.

Selection. After starting a new simulation run in the root of the tree, a child node is selected based on the so-called tree policy, and the selected node becomes the current node. This process is repeated until a leaf of the current tree is reached. Tree policy dictates which child node should be chosen for traversing next. It is crucial for the tree policy to balance between the exploitation of the most promising nodes (i.e. nodes with the biggest value of the expected reward) and exploration of other nodes, which might not have yet revealed their true potential.

Expansion. If the leaf that has been reached does not represent a final state, the tree is expanded by adding child nodes to the current leaf. Usually, all possible child nodes are added, and one of them is selected, possibly in a random way. In the future passes MCTS will prefer child nodes not yet visited or selected based on the tree policy (selection phase).

Simulation. If there are no further nodes to be visited at the current depth of the tree, the search process is simulated based on the so-called default policy, until the final state is reached and the value of the reward is known. The easiest way is to use a random simulation, although some simple, problem-specific rules may bring significant improvements. Using them may, however, decrease the simulation speed. During this step, no new nodes are added to the tree.

Backpropagation. The information about the received reward is backpropagated through all the nodes that were visited in the given simulation run, up until the root is reached and updated as well. Each node has to store at least information about the number of simulations that went through this node, and the average reward received from these simulations.

At the beginning, the search tree consists only of the root node. The tree is then built asymmetrically, as more promising nodes will be visited more often and new child nodes will be added in the more promising parts of the tree. Most researchers use a reward function, which assumes values from the [0, 1] range. The goal of MCTS is to find the decision, which maximizes the expected reward. It means that in the state described by the root node, the action leading to the root's child node maximizing the expected reward should be selected.

MCTS is well suited for problems in which the final state is guaranteed to be reached in a finite number of steps. It has several significant advantages. One of the most prominent ones is that there is no need to know any function to evaluate the intermediate states. This information is necessary only in the final state (e.g. after winning the game) and is then backpropagated through the nodes. This is unlike in the case of alpha-beta search. Also, MCTS can be stopped at any time and the current best solution can be given as a result. The general MCTS is presented as Algorithm 1. In the description, the term *terminal node* refers to the final state of the search (such as winning the game), and should not be confused with terminals in the Steiner problem.

The crucial part of the MCTS algorithm is the tree policy. The most widely applied approach is based on UCB proposed in [18] for a k-armed bandit problem. Each of the k arms of the bandit is defined by a random variable (interpreted as a reward), independent and identically distributed as the others, with an unknown expectation value. The goal is to develop (based on past rewards) a policy of selecting which bandit to play in order to maximize the reward. There is a straight-forward similarity to the problem of choosing a child node in MCTS. In [19] it was pro-

Algorithm 1: General MCTS

```
Input: root node root
Output: best child of root
while enough computational resources do
    leaf = SimulateWithTreePolicy(root)
    if leaf is not terminal node then
        new_node = Expand(leaf)
        reward = SimulateWithDefaultPolicy(new_node)
        UpdateTreeStatistics(new_node, root, reward)
    else
        reward = Reward(leaf)
        UpdateTreeStatistics(leaf, root, reward)
    end
end
return BestChild(root)
```

posed to use UCB1 (a version of UCB) as the tree policy. Combing MCTS and UCB1 used as the tree policy results in an algorithm named Upper Confidence Bound for Trees (UCT). In this algorithm, in the selection step of the MCTS procedure, being in the current node, an i-th child node is selected to maximize the UCT value given as:

$$\mathrm{UCT} = r_i + C\sqrt{\frac{2\ln n}{n_i}}, \tag{1}$$

where r_i is the average reward received in simulations going through i-th child node, n_i is the number of times i-th child node has been visited, n is the number of simulations going through the current (parent) node, and C is a constant value responsible for balancing between exploration and exploitation. In the case of UCT version of MCTS, reward value is expected to be within [0, 1]. From Eq. 1 it can be seen, that nodes with a higher average reward are preferred. However, their attractiveness decreases when they are visited many times (when n_i increases). It means that other nodes will be visited as well. In the special case when $n_i = 0$, this node is preferred before others. A significant finding of UCT is that given enough time and resources, the probability of selecting a suboptimal child node in the root converges to zero at a polynomial rate, with the number of simulations growing to infinity [19]. The UCT algorithm (i.e. Algorithm 1 with UCB1 used in "SimulateWithTreePolicy") will serve as a basis for the proposed algorithm for ESTP, as presented in Section 4.

3. Simple Heuristic for the Euclidean Steiner Tree Problem

In this section, the most important properties of Steiner trees is introduced and a measure of the quality of the candidate solutions is presented. Then, a special case of the problem for three terminal points is described. Based on its solution, a simple heuristic and a greedy search procedure are proposed to solve the ESTP for any number of terminals.

3.1. Basic Properties of Steiner Trees

Properties of Steiner trees have been studied for decades by many researchers. The readers interested in the detailed introduction are referred to [5]. Here, only the properties essential to the proposed algorithm are presented.

Property 1. Steiner minimal tree (SMT) for a set T of terminals has at most $p-2$ Steiner points. It is possible that no Steiner point (additional point) is necessary to connect the terminals with the minimum total length of edges. In this case, SMT equals the minimal spanning tree (MST) for the points in T.

Property 2. Degree property. In a Steiner tree, each Steiner point has exactly the degree of three (i.e. it is incident with exactly three edges).

Property 3. Angle property. Any two edges (from among three, see property 2), incident with the Steiner point, make exactly $120°$ with each other.

Figure 1 presents a minimal spanning tree and a Steiner tree. It can be observed that properties 1–3 are satisfied. It should be mentioned that a given set of Steiner points that satisfy the properties above, does not necessarily constitute an optimal solution. Properties 1–3 are the necessary conditions for a set of candidate Steiner points. It is possible to build a Steiner tree which satisfies the three properties and yet it is not the optimal tree.
The common criterion to compare the candidate solutions in ESTP is to calculate the so-called Steiner ratio for each candidate Steiner tree. The general procedure is as follows. Having a set of terminals T and a set of candidate Steiner points S, let $P = T \cup S$. Build the minimum spanning trees for T and for P. Then, the Steiner ratio is defined as:

$$\rho = \frac{L(MST_P)}{L(MST_T)}, \tag{2}$$

where $L(.)$ is the total sum of lengths of edges from the corresponding tree, MST_T and MST_P are the minimal spanning trees for the terminals and a set of points created by combining terminals with Steiner points, respectively. This criterion is minimized for the solution of the considered problem of finding the minimum Steiner tree. However, it can be calculated for any set of points S, even points which do not satisfy the required properties. For that reason, Steiner ratio is a useful criterion that can be used in any heuristic search algorithm.

3.2. Fermat Point

The task of finding the minimal Steiner tree was first introduced by Fermat as the problem of finding a point with a minimum distance from other three points that are given. Later the problem has been generalized by allowing any number of given points (terminals T) and any number of additional points (Steiner points S), while the task was to interconnect all terminals, achieving the minimum total length of all edges. The original question formulated by

Fermat can be given as follows. For a given set of three points on the Euclidean plane, A, B, and C, find point F (called Fermat point), such as for any other point F' the condition is satisfied:

$$|F'A| + |F'B| + |F'C| > |FA| + |FB| + |FC| . \qquad (3)$$

This is, in fact, ESTP for a set of three terminals, and it is known from the property 1, that at most one Steiner point can be present in the optimal solution. This problem can be solved easily. Two situations are possible. If points A, B and C form a triangle in which all internal angles are smaller than 120°, then the only Steiner point is located at the Fermat point F of the triangle. The SMT is equivalent to the minimum spanning tree (MST) including three triangle points and the Fermat point. In the second case, if there is an internal angle equal to or greater than 120°, the Steiner point is incident with the triangle vertex (A, B or C) in which the two sides of the triangle meet at 120° degrees or more. The Steiner minimum tree (and the solution to the original Fermat problem at the same time) is the MST spanned on the three triangle points. The procedure of finding the Fermat point can be formulated in several ways. One of them can be given as follows.

Given three points A, B, and C on the Euclidean plane, if $\measuredangle ABC \geq 120°$ or $\measuredangle BAC \geq 120°$ or $\measuredangle ACB \geq 120°$, then return B, A or C as the Fermat point, respectively. Otherwise, select any two sides of the triangle ABC. Construct an equilateral triangle on each of the chosen sides. Find two lines, going through each new vertex of the constructed triangles and their opposite vertices of the original triangle. The Fermat point F is the point of intersection of the two lines.

3.3. Greedy Heuristic for ESTP

It can be observed that the Steiner tree is inherently connected with the procedure of building the MST. If the set of Steiner points is given, the Steiner tree can be found by calling one of the efficient algorithms for MST construction, such as the Prim algorithm [4]. However, this section proposes a simple heuristic, which considers the problem from another point of view.

The primary switch is to start the Prim algorithm without having the set of Steiner points given. Instead, the candidate Steiner points are added dynamically during the construction of the MST. The resulting procedure can be considered as a greedy optimization procedure. The main idea is based on the Prim algorithm. The Prim algorithm starts by selecting any of the given points. Then, in each step, a new point is added to the growing tree as the one with the minimum distance to any of the previously selected points. This step is repeated as long as all of the points are selected.

The proposed modification is as follows. The first two points are selected as in the original Prim algorithm. However, next steps are modified. A new point p is added to the growing MST by directly connecting it to its nearest neighbor p' or by connecting it through f (a Fermat point) to two points p' and p''. The choice is made in a greedy way, based on the total sum of lengths of edges in the partial tree. The formal description is given as Algorithm 2. The meaning of the correction steps is explained further in this section. In the algorithm, $|e|$ is the length of the corresponding edge e.

Algorithm 2: Greedy algorithm for Steiner tree

```
Input: Set of points $T$ (terminals), $correct\_all$ (boolean value)
Output: Steiner tree $ST$ for points in $T$, set of Steiner points $SP$
1. Set $E = \emptyset$ ($E$ is the set of edges in $ST$), $SP = \emptyset$
2. Select any point $p$ from $T$. Set $T' = \{p\}$ and $T = T \setminus \{p\}$
3. Select point $p$ from $T$ with the minimum distance to the point $p'$ in $T'$. Set $T' = T' \cup \{p\}$ and $T = T \setminus \{p\}$. Add the edge $(p, p')$ to $E$.
4. Select point $p$ from $T$ with the minimum distance to any point $p'$ in $T'$.
5. Select any point $p''$ from $T'$, so that $p'' \neq p'$ and $(p', p'') \in E$.
6. Calculate Fermat point $f$ for the three points $p$, $p'$ and $p''$.
7. if $f$ is a valid Fermat point (i.e., not incident with $p$, $p'$ or $p''$) and $|(p,p')| + |(p',p'')| > |(p,f)| + |(p',f)| + |(p'',f)|$ then
    remove $(p',p'')$ from $E$,
    add $(p,f)$, $(p',f)$ and $(p'',f)$ to $E$,
    set $T = T \setminus \{p\}$, set $T' = T' \cup \{p,f\}$, add $f$ to $SP$,
else
    add $(p,p')$ to $E$, set $T = T \setminus \{p\}$, set $T' = T' \cup \{p\}$
end
8. If $correct\_all = true$, correct solution (Algorithm 3)
9. Repeat from step 4 until $T \neq \emptyset$.
10. Correct solution (Algorithm 3).
11. Return $ST(T',E)$ and $SP$
```

3.4. Correction Procedure

After adding a new Fermat point as a Steiner point in Algorithm 2, some of the previously added Steiner points may no longer satisfy angle- and degree-related conditions required from all Steiner points. One example is presented in Fig. 2. Steiner point $F1$, inserted by the algorithm, satisfies the angle and degree properties, as it was calculated as the Fermat point for the three connected terminals (Fig. 2, step A). After the second Steiner point $F2$ is inserted (step B) as the Fermat point calculated for three points connected thereto, $F1$ no longer satisfies the angle condition. The situation is fixed using the proposed correction procedure, which is depicted in part C of Fig. 2. Note that during the correction procedure point $F2$ is also moved from its original position. The correction procedure is given as Algorithm 3. The main idea is to perform two actions repeatedly: 1) to remove all candidate

```
Algorithm 3: Correction procedure
Input: Set of points $T$ (terminals), set of Steiner points
       $SP$ (some of them can be invalid), $\varepsilon$ – maximum
       allowed violation of angle condition for Steiner
       points.
Output: $SP$ – set of valid Steiner points (i.e., satisfying
        the angle and degree conditions)
1. Let $P = T \cup P$ and $MST_P$ be the minimum spanning
tree for $P$.
2. Identify Steiner points in $SP$ which have degrees not
equal 3 in $MST_P$ and remove them from $SP$; recalculate
$MST_P$.
3. Repeat step 2 until no $sp$ is removed in step 2.
4. Identify Steiner point $sp$ in $SP$ which violates the angle
condition the most in $MST_P$. If this violation is more than
$\varepsilon$, recalculate this point as a Fermat point of the triangle
formed by the three points connected to $sp$ (if the Fermat
point is not valid, remove $sp$); recalculate $MST_P$.
5. If $SP$ has been modified in step 4, go to step 2.
6. Return $SP$.
```

Steiner points which do not satisfy the degree-related condition, and 2) to recalculate the coordinates of those Steiner points which do not meet the angle-related condition (the recalculation is done by means of calculating new Fermat points). These two steps are repeated until no change is necessary.

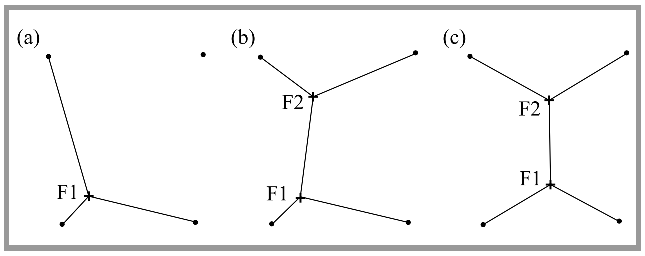

Fig. 2. Effect of the correction procedure.

A parameter *correct_all* is introduced in Algorithm 2 to limit the computational cost of the correction procedure. Only when it is set to *true*, the correction procedure will be called after adding each new Steiner point. Otherwise, it is called only at the end of the algorithm, after connecting all terminals. As revealed by tests (see Section 5), in practice, *correct_all* can be in most situations set to *false* without decreasing the quality of the final solution significantly.

Algorithm 2 together with Algorithm 3 clearly constitute a greedy search procedure, as in step 7 of Algorithm 2, always locally better (shorter) connections are selected. This does not guarantee a global optimum. However, it can provide a feasible solution very fast. In fact, it is possible to run Algorithm 2 starting with each possible terminal as the starting point in this greedy procedure (point p selected in step 2 of Algorithm 2). Then, the best solution from all runs is chosen as the final solution. This simple greedy heuristic can be further extended by allowing not optimal local choices in step 7 of Algorithm 2.

4. Monte Carlo Tree Search for Euclidean Steiner Tree Problem

Algorithm 2 shares a common drawback with all greedy optimization algorithms. It may be trapped by local minima, as the choices that are optimal locally do not always lead to the global optimum. This section proposes two ways to relax the greediness and allow not optimal local choices with the hope to improve the solution.

4.1. Pure Monte Carlo Simulations for ESTP

The easiest way to relax the greediness of Algorithm 2 is to allow a random selection of one of the two options in step 7. The modified procedure is as follows. If the Fermat point f is not valid, connect the new point as in the original Prim algorithm; else, if the Fermat point is valid, randomly select (with equal probabilities) whether the new point is connected as in the original Prim algorithm, or through the Fermat point. After the last terminal point is connected, the found solution (Steiner tree and Steiner points) is remembered together with its Steiner ratio. The whole procedure is repeated for a given number of iterations. After each iteration, it is checked whether a better solution is found. Thus, each iteration is independent of others and can be easily parallelized. The proposed Monte Carlo simulation for ESTP is given as Algorithm 4. To simplify the notation, it is assumed that the set of terminals T is recovered to its original input state at the beginning of each iteration of the f or loop. Function RANDOM(0, 1) returns a random value from a uniform distribution from the [0, 1) range.

4.2. Hybridization of Prim Algorithm and MCTS

The drawback of Algorithm 4 is that the path leading to the solution (i.e. how the points were connected to the growing tree) is neither remembered nor used in subsequent iterations. In this section, it is proposed how the quality information about previous simulations can be utilized. The presented approach can be viewed as hybridization of the Prim algorithm and MCTS. It is important to clarify some potential ambiguities here. In fact, there are two trees in the proposed approach. The first one is the Steiner tree, which is the required solution of the problem at hand. During each iteration of Algorithm 4, a new Steiner tree is built and evaluated. The other tree in the proposed Algorithm 5 is the MCTS tree, which is a data structure used to remember the statistics about choices from previous iterations and to guide the search process.

To describe the proposed MCTS algorithm, one has to define the tree policy, the default policy, and the reward function, expanding the node and updating the statistics of the node. First, let us describe the proposed MCTS algorithm for ESTP in general. The simulation starts with only the root node in the MCTS tree. Expansion of the root node

Algorithm 4: Pure Monte Carlo simulation for Steiner tree

```
Input: Set of points $T$ (terminals), $correct\_all$, $number\_of\_simulations$
Output: Steiner tree $ST_{best}$ for points in $T$, set of Steiner points $SP_{best}$ corresponding to $ST_{best}$
1. $ST_{best} = MST_T$, $SP_{best} = \emptyset$, $\rho_{best} = 1$ (Steiner ratio)
2. for $i = 0$ to $number\_of\_simulations$ do
    a. Set $E = \emptyset$ ($E$ is the set of edges in $ST$), $SP = \emptyset$
    b. Select any point $p \in T$. Set $T' = \{p\}$, $T = T \setminus \{p\}$
    c. Select point $p \in T$ with the minimum distance to the point $p' \in T'$. Set $T' = T' \cup \{p\}$ and $T = T \setminus \{p\}$. Add the edge $(p, p')$ to $E$
    d. Select point $p \in T$ with the minimum distance to any point $p' \in T'$.
    e. Select any point $p'' \in T'$, so that $p'' \neq p'$ and $(p', p'') \in E$.
    f. Calculate Fermat point $f$ for the three points $p$, $p'$ and $p''$.
    g. if $f$ is a valid Fermat point (i.e., not incident with $p$, $p'$ or $p''$) AND RANDOM(0,1) $< 0.5$ then
        remove $(p', p'')$ from $E$,
        add $(p,f)$, $(p',f)$ and $(p'',f)$ to $E$,
        set $T = T \setminus \{p\}$, set $T' = T' \cup \{p, f\}$, add $f$ to $SP$,
    else
        add $(p, p')$ to $E$, set $T = T \setminus \{p\}$, set $T' = T' \cup \{p\}$
    end
    h. if $correct\_all = true$ then
        correct solution (Algorithm 3)
    end
    i. Repeat from step 2.d until $T \neq \emptyset$.
    j. Correct solution (Algorithm 3).
    k. $\rho = L(MST_{T \cup SP})/L(MST_T)$
    l. if $\rho < \rho_{best}$ then
        $ST_{best} = MST_{T \cup SP}$, $SP_{best} = SP$, $\rho_{best} = \rho$
    end
end
3. Return $ST_{best}$ and $SP_{best}$
```

is performed by selecting the first terminal as in step 2b in Algorithm 4. Thus, the number of branches from the root node equals the number of terminals. On all following levels of the MCTS tree, expanding the node creates two branches at the most, one is for direct connection of the new terminal (as in the Prim algorithm), while the other for connection through the newly created Steiner point. In these cases in which the Steiner point cannot be created as a valid Fermat point, only one branch is created. While descending from the root, at each existing node, one of the two possible branches is selected based on the accumulated statistics in that node according to the common tree policy based on UCT (Eq. 1) described in Section 2. Thus, the process is performed differently than in step 2g of Algorithm 4. On the other hand, the default policy of the proposed MCST for ESTP is done exactly as in step 2g of Algorithm 4. It means, that when there are no statistics available on the current level of the MCTS tree, the new terminal is connected to the growing Steiner tree directly

Algorithm 5: UCT for Euclidean Steiner tree problem

```
Input: Set of points $T$ (terminals), $correct\_all$, $number\_of\_simulations$
Output: Steiner tree $ST_{best}$ for points in $T$, set of Steiner points $SP_{best}$
1. $ST_{best} = MST_T$, $SP_{best} = \emptyset$, $\rho_{best} = 1$ (Steiner ratio of the best Steiner tree found), $root = \emptyset$
2. for $i = 0$ to $number\_of\_simulations$ do
    a. Set $E = \emptyset$ ($E$ is the set of edges in $ST$), $SP = \emptyset$
    Tree policy
    b. If $root = \emptyset$, expand $root$, the number of child nodes equals the number of terminals $|T|$. Each child node is associated with one terminal.
    c. Select $current\_node$ as the child node of the $root$ maximizing its $UCT$ value, giving priority to those child nodes which have not been visited yet. Set $T' = \{p\}$ and $T = T \setminus \{p\}$, where $p$ is the terminal associated with $current\_node$.
    d. Repeat until $current\_node$ is not a leaf in MCTS tree: Select $new\_node$ as the child node of $current\_node$ which maximizes its $UCT$ value, giving priority to child nodes not yet visited. Let $p$ be the terminal connected to the growing Steiner tree in a way described by node $new\_node$. If $new\_node$ describes direct connection, set $T' = T' \cup \{p\}$ and $T = T \setminus \{p\}$, add the edge $(p, p')$ to $E$, where $p'$ is the point from $T'$ closest to $p$ (remembered in $new\_node$); Else If $new\_node$ describes connection through Fermat point $f$, remove $(p', p'')$ from $E$, add $(p, f)$, $(p', f)$ and $(p'', f)$ to $E$, set $T = T \setminus \{p\}$, set $T' = T' \cup \{p, f\}$, add $f$ to $SP$, where $p''$ has the same interpretation as in Algorithm 4, and is remembered in $new\_node$ together with $p'$. Set $current\_node = new\_node$. Correct solution if $correct\_all$ is $true$.
    Expanding
    e. If $current\_node$ is a leaf in MCTS tree, expand $current\_node$ as follows. Find point $p$ in $T$ with the closest distance to any point $p'$ in $T'$. Select any point $p''$ from $T'$, so that $p'' \neq p'$ and $(p', p'') \in E$. Create two child nodes of $current\_node$ in MCTS tree, one for the direct connection of $p$ to $p'$, the second for the case when $p$ is connected to $p'$ and $p''$ through $f$ (Fermat point for points $p$, $p'$ and $p''$), if $f$ is valid.
    Default policy
    f. Execute steps 2d–2l from Algorithm 4 (it can update $ST_{best}$ and $SP_{best}$).
    Update MCTS tree statistics
    g. Calculate reward value based on the Steiner ratio of the Steiner tree that has been built during this iteration: $reward = RewardFunction(\rho)$
    h. Update $UCT$ value of each node in MCTS tree that has been visited during this iteration (according to Eq. 1).
end
3. Return $ST_{best}$ and $SP_{best}$
```

or through a new Steiner point, as long as there is a valid Fermat point. The choice is made randomly with the probability of selecting either option equaling 0.5. Each iteration ends when the last terminal is connected to the Steiner tree being built during this iteration. The Steiner ratio is calculated, and its value can be used as the reward. However, it needs to be converted, as the original UCT algorithm requires the reward to be in the range of [0, 1]. Several possible ways of performing this conversion are described in the next section. Having the reward in the proper form makes it possible to use it for updating the UCT value of nodes in the MCTS tree according to Eq. 1. The proposed MCTS for ESTP is described more formally as Algorithm 5.

4.3. Reward Functions

The role of the reward functions proposed here is to provide a mapping from the Steiner ratios to the reward values that can be used in MCTS. In ESTP, the Steiner trees are evaluated using the Steiner ratio ρ, while in MCTS, there is a need to assign reward values for the leafs which are from the range of [0, 1]. Additional steps must be taken to satisfy this requirement. For the minimum spanning tree for the set of terminals, the Steiner ratio equals 1. Having any feasible solution of ESTP means that the Steiner ratio $\rho < 1$. It was proved [5], that for problems in the 2D Euclidean space, the minimum possible value of ρ is $\rho_{opt} = \frac{\sqrt{3}}{2} \approx 0.866$. Thus, is possible to define a mapping from the calculated ρ to the reward value that can be used in the UCT algorithm. The main idea is to assign high rewards (close to 1) for solutions with ρ close to ρ_{opt}, and low rewards (close to 0) for solutions with ρ close to 1. Thus, it is enough to define the reward functions on the domain $[\rho_{opt}, 1]$ as $reward(\rho) : [\frac{\sqrt{3}}{2}, 1] \rightarrow [0,\ 1]$. The proposed reward functions are as follows:

- $reward_{linear}(\rho) = \frac{1-\rho}{a}$ with example value of parameter $a = 7.45$, $reward_{Gauss}(\rho) = e^{-\frac{(\rho_{opt}-\rho)^2}{\sigma}}$ with example value of parameter $\sigma = 0.005$,
- $reward_{sigmoidal}(\rho) = 2 - \frac{2}{1+e^{-\beta(\rho-\rho_{opt})}}$ with example value of parameter $\beta = 40$,
- $reward_{log}(\rho) = 1 - a_2 \ln(1 + a_1(\rho - \rho_{opt}))$ with example value of parameters $a_1 = 1000$ and $a_2 = 0.204$,
- $reward_{cos}(\rho) = \cos\left(c(\rho - \rho_{opt})\right)$ with example value of parameter $c = 11.725$.

These functions differ in how they promote the decrease in the Steiner ratio. For example, the cosine reward function adds the reward faster for Steiner ratios close to 1 than it is the case with the linear function. The usefulness of these reward functions is verified, using numerical experiments, in the next section.

5. Experimental Studies

In this section, the results of the proposed algorithms applied to problems of various sizes are presented. Randomly generated problems, as well as problems with known optimal solutions from OR-Lib [15], are considered. The Friedman test (with Shaffer post hoc tests) is used, to compare the results. The testing procedure is as follows. Only the average results for each problem of each algorithm are considered by the Friedman test. All problems of a given size and all algorithms are taken up at once to discover potential differences among the average behaviors of the algorithms. Then, if any differences are found, a set of pair-wise post-hoc comparisons between algorithms is calculated.
The Friedman test is recommended where one wants to compare the general behavior of more than two algorithms on several test problems. This two-step procedure is regarded as a better approach than performing many pair-wise comparisons, such as the Wilcoxon test, which would increase the overall probability of making at least one type I error, which would erroneously discover differences between algorithms which, in fact, perform equally in general. More details about the testing procedure adopted can be found in [20]. All the tests were calculated using the Keel software package [21].

5.1. Results for Random Problems

Preliminary tests. In regard to random problems, for each problem size, twenty random problems are generated from a uniform distribution on $[0,1]^2$. Each algorithm is run ten times on each problem. In the Friedman test, only the mean values of Steiner ratios are compared.
In the presentation of the results the following abbreviations are adopted:

- greedy1 – Algorithm 2 run $|T|$-times using each terminal point as the starting point in step 2 in Algorithm 2 and the parameter *correct_all* set to *true*;
- greedy2 – the same as algorithm greedy1 except the parameter *correct_all* is set to *false*;
- MC10k, MC100k – pure Monte Carlo simulation for Steiner tree (Algorithm 4) run with 10,000 and 100,000 iterations, respectively,
- UCT10k and UCT100k – UCT algorithm (Algorithm 5) with the linear reward function run with 10,000 and 100,000 iterations, respectively.

In the preliminary tests, the algorithms have been first applied to random problems with 20, 30, 40 and 50 terminals. The goal was to check how the ideas introduced scale with the number of terminals. Only the linear reward function is used and the $C = 1$ in Eq. 1. For the sake of space, only the conclusions, and not the detailed results, are provided here. For problems with 20 terminals, the *p-value* calculated from the Friedman test was 0.884. It means that the test does not discover any significant differences among the algorithms. The results of all algorithms are similar. For problems with 30 terminals, UCT and MC algorithms with 100k iterations were ranked as best. The *p-value* from the Friedman test was 0.00042, which means that significant differences exist in the performance of the algorithms and Shaffer's post hoc procedures were calculated. They discovered significant differences between UCT100k and MC100k algorithms and UCT10k and MCT10k algorithms showing that 10k iterations are not enough for MCT and UCT algorithms. However, no significant difference was discovered between MCT100k and UCT10k, which can be interpreted as the first sign of the advantage of UCT over MC. Additionally, there were no significant differences discovered between any of the greedy and the winning UCT100k and MC100k algorithms.

Results for problems with 40 and 50 terminals showed that the Friedman *p-values* keep decreasing, demonstrating that the differences are more clear as the problems become more difficult. The proposed UCT algorithm with 100k iterations was always ranked as the best. However, considering Shaffer post-host comparisons, significant differences between UCT and MC algorithms are not discovered when given the same number of iterations (10k or 100k). The important difference between the winning UCT100k and the greedy 2 algorithm was signaled only once. On the other hand, the greedy algorithms perform significantly better than MC10k and UCT10k. It can be summarized that the majority of differences discovered in the average behavior of the algorithms stem from the poor performance of MC10k and UCT10k algorithms. Another important observation is that the greedy2 algorithm has never been significantly worse than greedy1. Based on this, in the remaining experiments, partial solutions found after each iteration of MC and UCT algorithms are also not corrected, and only the final solution is corrected.

Scaling the reward function and checking the influence of parameter C (50 terminals). There are two important parameters for UCT that may influence the performance of UCT, the parameter C from Eq. (1) and the reward function. C is responsible for tuning the importance of exploration vs. exploitation, thus setting it correctly is crucial. A set of 20 random problems with 50 terminals is considered here.

In this set of experiments, only the greedy1 algorithm is considered. All UCT algorithms and the MC algorithm were allowed to iterate over 100k iterations. The value of C is given after the underscore, i.e. $UCT\_C0.5$ means that the UCT algorithm with the linear reward function has been run with $C = 0.5$. The algorithm abbreviated UCT in the following results is the algorithm used in the previous set of experiments ($C = 1$ and linear reward function). However, algorithms $UCT\_C1.0$, $UCT\_C2.0$, $UCT\_C0.5$ and $UCT\_C0.1$ have been further modified by scaling the linear reward function. It has been observed that the best solutions found had never had the Steiner ratio lower than 0.96. The theoretical minimum for the Steiner ratio is $\rho_{opt} = \frac{\sqrt{3}}{2} \approx 0.866$, it is, however, only achieved for some special cases of regular grids of terminals. In the case of real life problems, the Steiner ratio for a given problem would be higher. In such cases, the reward function introduced and defined over the range $[\rho_{opt}, 1]$, does not use the full range of the domain. The idea here is to limit the domain of the reward function. The linear reward function has been scaled, i.e. its domain is only [0.96, 1] and the parameter $a = 25$.

The Friedman ranks presented in Table 1, and the small *p-value* shows that, in fact, the setting of parameter C has a huge impact on the performance of the proposed UCT algorithm. The pair-wise comparisons revealed that by setting C to a proper value, the UCT algorithm performs significantly better than MC. Also, it can be observed that setting $C = 2$ significantly decreases the performance of the UCT algorithm. There were no significant differences discovered between the best-performing algorithms ($UCT\_C0.5$ and $UCT\_C0.1$) and the greedy algorithm. However, when the numbers of best solutions found are considered (showing the ability of the algorithms to find best-known solutions for the problems), it can be stated that the UCT algorithm with the proper value of C should be preferable. The influence of the scaling of the reward function is not clear from these experiments. On the one hand, the UCT version abbreviated simply as UCT (no scaling, $C = 1$) is ranked lower than any of the modified UCTs, except for the UCT with $C = 2$. On the other hand, no significant differences between UCT and any of the modified UCT algorithms have been discovered. To further investigate this issue, additional experiments have been calculated, as presented in the later part of this section.

Autoscaling the reward function (problems with 70 terminals). In this set of experiments, 20 random problems with 70 terminals are considered. Here we test the influence of the different reward functions introduced earlier. The number of MC and UCT algorithm iterations exceeded 150,000. All UCT algorithms use parameter $C = 0.1$. Algorithms abbreviated as *lin*, *cos*, *Gauss*, *log*, and *sigm* are the algorithms with reward functions described in Subsection 4.3. Algorithms with suffix *_sc* in their name are the algorithms with the scaled versions of the functions. The scaling procedure is more elaborate here. Instead of assuming one new lower limit for the Steiner ratio, the idea is to scale the reward functions based on the Steiner ratio of the best-known solution for each problem separately. In practice, the *greedy*1 algorithm is run as the first one for a given problem, and the Steiner ratio ρ_{greedy} of the solution found by this algorithm is used to calculate the new lower limit for the domain of the reward function used in UCT algorithms. In fact, ρ'_{opt} is calculated as $\rho'_{opt} = 0.995 \cdot \rho_{greedy}$. Then ρ'_{opt} is used to define the new scaled versions of the reward functions by calculating proper parameter values. The parameters of the scaled versions of the reward functions are calculated as follows:

- $a = \frac{1}{1-\rho'_{opt}}$ for the linear function,
- $\sigma = \frac{-(\rho'_{opt}-1)^2}{\ln 0.04}$ for the Gaussian function,
- $\beta = \frac{\ln(1/0.98-1)}{\rho'_{opt}-1}$ for the sigmoidal function,

Table 1
Friedman ranks and numbers of best solutions found for problems with 50 terminals. Entries marked in bold indicate algorithms which performed significantly worse than the best performing one; *p-value* $= 0.00225$

Algorithm	UCT_C0.5	UCT_C0.1	greedy 1	UCT_C1.0	UCT	UCT_C2.0	MC
Ranking	2.6	3.35	3.9	3.9	4.25	**4.6**	**5.4**
Best solutions	9	10	3	7	9	7	3

Table 2
Friedman ranks and numbers of best solutions found for problems with 70 terminals. Entries marked in bold indicate algorithms which performed significantly worse than the best performing one; *p-value* $= 6 \cdot 10^{-11}$

Algorithm	lin	lin_sc	cos	Gauss	log_sc	greedy 1	log	sigm_sc	cos_sc	sigm	MC	Gauss_sc
Ranking	2.8	3.2	3.65	4.05	5.55	**7.5**	**7.55**	**7.55**	**8.6**	**9**	**9.2**	**9.35**
Best solutions	7	7	7	6	3	4	2	2	3	0	1	3

- $a_2 = \frac{0.96}{\ln(1+(1-\rho'_{opt})a_1)}$,
- $a_1 = 1000$ for logarithm based function, $c = \frac{\pi}{2}(1 - \rho'_{opt})$ for cosine function.

These derivations have been based on the assumptions, that the respective reward function should have values close to 0 and 1 at the limits of its domain.

Friedman ranks presented in Table 2 show that there are significant differences among the algorithms with different reward functions. Several important observations can be made based on pair-wise comparisons. MC and greedy algorithms are outperformed by the UCT algorithm, provided that the reward function is properly selected. UCT with linear, linear scaled, and cosine reward functions are significantly better than greedy and MC algorithms. UCT with the Gaussian reward function is also significantly better than MC. The superiority of UCT with these functions is also visible when considering the numbers of best solutions found.

On the other hand, MC and greedy algorithms are not outperformed by all of the UCT configurations, which shows that the proper choice of the reward function is crucial. At this point, a simple linear function seems to be the best option. Not only does it perform best, but it is also faster to calculate than the nonlinear functions. Surprisingly, scaling is never beneficial. Even if the scaled version of UCT is ranked higher than the original version, the difference is never significant. To explain this, let us ask the question about how fast a given algorithm finds the best solution in a given run (i.e. in which iteration). The statistics over all runs on all 20 problems with 70 terminals have been collected. It was observed, that UCTs based on scaled reward functions tend to converge prematurely, reaching the final solution early.

All results presented thus far can be summarized as follows. The proposed UCT algorithm depends on the proper choice of the parameter C and the reward function. When properly set, the proposed UCT algorithm performs significantly better than the greedy heuristic and the simple MC simulation. On the other hand, when not configured correctly, UCT may perform poorly. A good choice for C is 0.1, and a simple linear reward function can be recommended. The superiority of the UCT algorithm is becoming apparent when the dificulty of the considered ESTP grows.

5.2. Results for OR-Lib Problems

In this section, it is shown that the proposed algorithm is, in fact, able to find good quality solutions for ESTP. A set of benchmark problems from the OR-Lib library is used [15]. The optimal solutions for these problems are known and can be used to assess the results achieved by the proposed techniques. In the tests, the following files from OR-Lib have been used: estein10, estein20, estein30, estein40, estein50, estein60, estein70, estein80, estein90 and estein100. Those files have been taken from http://people.brunel.ac.uk/∼mastjjb/jeb/orlib/esteininfo.html. Each file contains 15 problems with the number of terminals corresponding to the number in the name of the file. The greedy algorithm (greedy1), MC and several UCT algorithms have been considered. The number of iterations in MC and UCT algorithms has been set to 250k. For UCT algorithms, $C = 0.1$. All reward functions have been used, however, without scaling.

All UCT and MC algorithms have been run ten times for each problem. Friedman ranks have been calculated as previously. The results obtained by the algorithms have been compared with the optimal solutions taken from the OR-Lib library.

For each algorithm, and for each problem, the best result from among ten runs is selected as the Steiner ratio ρ_{best}. Then, the relative error is computed as

$$err = \frac{\rho_{best} - \rho_{opt}}{\rho_{opt}} \cdot 100\% ,$$

where ρ_{opt} is the Steiner ratio of the optimal solution for a given problem, taken from OR-Lib. For each algorithm, such errors were calculated for each of the 15 problems from the group of problems of a given size. The minimum, maximum, mean and standard deviation of these relative error values have been calculated. The whole procedure has been repeated for each file from OR-Lib,

Table 3
Friedman ranks for problems from OR-Lib with 90 terminals; *p-value* $= 6.4 \cdot 10^{-9}$

Algorithm	UCT_linear	UCT_cos	UCT_gauss	greedy 1	UCT_log	UCT_sigm	MC
Ranking	1.6	2.7333	3.2	**4.2667**	**4.5333**	**5.4667**	**6.2**

Table 4
Friedman ranks for problems from OR-Lib with 100 terminals; *p-value* $= 1.03 \cdot 10^{-6}$

Algorithm	UCT_lin	UCT_cos	greedy1	UCT_gauss	UCT_log	UCT_sigm	MC
Ranking	2.1333	2.6667	3.6	3.6	**4.5333**	**5.6667**	**5.8**

i.e. each group of 15 problems of a given size. For the sake of space, detailed Friedman ranks for problems with 90 and 100 terminals are only presented in Tables 3 and 4. Entries marked in bold indicate algorithms which performed significantly worse than the best performing one. For problems of all sizes only the summary and conclusions are given below.

The proposed algorithms can provide near optimal solutions for known problems from the OR-Lib library. On average, the best solutions found by the algorithms are never by more than 1% worse than the optimal ones, and UCT algorithms have shown to be able to provide the best results. Starting with the problems with 20 terminals, in the worst cases the best solutions found by the algorithms were never by more than 2% worse than the optimal ones (except for 2.2% for the greedy1 algorithm in the case of size 80 problems). In the case of problems with ten terminals, those errors exceeded 2% (they were lower than 3%) due to the existence of one EST problem which turned out to be difficult for all the algorithms. For all OR-Lib problems, Friedman ranks were consistent with the results from the previous section. For smaller problems, the average performance is not distinguishable. When the number of terminals increases, the UCT algorithm with a proper reward function can provide better results on average. For problems with ten terminals, all algorithms performed similarly, according to the Friedman test. For problems with 20 terminals, the Friedman test discovered significant differences (*p-value* $= 0.0174$), however, post-hoc procedures were not able to detect the sources of the differences.

For bigger problems (from 30 terminals up), significant differences were discovered. There are two sources of the differences. First, when the number of terminals grows, MC and greedy algorithms are consistently ranked lower than an instance of the UCT algorithm. The differences are not always discovered as significant, according to Shaffer's post-hoc procedures, showing that the proposed greedy approach is also interesting on its own. It loses, however, when the best solutions found are analyzed. The second source of the differences among algorithms is the selection of the reward function for the UCT algorithm. The results confirm that the choice of the reward function is crucial and there is no clear winner. The linear reward function is a good choice for problems with a bigger number of terminals. On the other hand, it performed worse for smaller problems. For example, for problems with 30 terminals it performed significantly worse than UCT with the sigmoidal reward function. The cosine function performed poorly for smaller problems, on the other hand, it provided, on average, the best solutions for problems with 100 terminals. This justifies the use of different reward functions. However, the linear reward function proves to be a good choice as computational costs are lower than for other reward functions.

Due to the limitations of this publication, a more detailed elaboration of the results is available from the author upon request.

6. Conclusions

A novel MCTS algorithm for ESTP on a plane has been proposed. A simple greedy heuristic has been introduced based on the Prim algorithm and the calculation of the Fermat point. It was utilized in MC simulations and the UCT-based algorithm, the most popular version of MCTS. To the best of the author's knowledge, it is the first time when the MCTS algorithm has been adapted for ESTP. Several reward functions have been proposed for use with UCT. The numerical experiments showed a high importance of the proper selection of the reward function. The proposed UCT algorithm can provide better results than simple MC simulations and greedy algorithms tested. It was shown, based on the benchmark problems from OR-Lib, that the proposed algorithms could provide near optimal solutions. The future research direction is to adapt the proposed MCTS to problems with a higher number of dimensions.

References

[1] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search", *Nature*, vol. 529, pp. 484–489, 2016.

[2] C. Browne and E. Powley, "A survey of Monte Carlo tree search methods", *IEEE Trans. on Intell. and AI in Games*, vol. 4, no. 1, pp. 1–49, 2012.

[3] A. Auger and O. Teytaud, "Continuous lunches are free plus the design of optimal optimization algorithms", *Algorithmica*, vol. 57, no. 1, pp. 121–146, 2010.

[4] R. C. Prim, "Shortest connection networks and some generalizations", *The Bell System Tech. J.*, vol. 36, pp. 1389–1401, no. 6, 1957.

[5] M. Brazil and M. Zachariasen, *Optimal Interconnection Trees in the Plane. Theory, Algorithms and Applications*, 1st ed. Springer, 2015.

[6] E. Miguez, J. Cidras, E. Diaz-Dorado, and J. L. Garcia-Dornelas, "An Improved Branch Exchange Algorithm for Large Scale Distribution Network Planning", *IEEE Power Engineering Review*, vol. 22, pp. 58–58, 2002.

[7] M. M. M. Brazil, C. C. Ras, and D. D. Da Thomas, "Relay augmentation for lifetime extension of wireless sensor networks", *IET Wirel. Sensor Sys.*, pp. 1–29, 2013.

[8] R. P. Mondaini and N. V. Oliveira, "Intramolecular structure of proteins as driven by Steiner optimization problems", in *Proc. 18th Int. Conf. on Syst. Engin. ICSEng 2005*, Las Vegas, NV, USA, 2005, vol. 1, pp. 490–491.

[9] Z. A. Melzak, "On the problem of Steiner", *Canadian Mathem. Bull.*, vol. 4, pp. 143–148, 1961.

[10] P. Winter and M. Zachariasen, "Euclidean Steiner minimum trees: An improved exact algorithm", *Networks*, vol. 30, no. 3, pp. 149–166, 1997.

[11] D. Trietsch and F. Hwang, "An improved algorithm for Steiner trees", *SIAM J. on Appl. Mathem.*, vol. 50, no. 1, pp. 244–263, 1990.

[12] W. D. Smith, "How to find Steiner minimal trees in euclideand-space", *Algorithmica*, vol. 7, no. 1–6, pp. 137–177, 1992.

[13] V. L. do Forte, F. M. T. Montenegro, J. A. de M. Brito, and N. Maculan, "Iterated local search algorithms for the Euclidean Steiner tree problem in n dimensions", *Int. Trans. in Oper. Res.*, vol. 23, no. 6, pp. 1185–1199, 2015.

[14] J. Barreiros, "A hierarchic genetic algorithm for computing (near) optimal Euclidean Steiner trees", in *GECCO 2003: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference*, A. M. Barry, Ed. Chigaco: AAAI, 2003, pp. 56–65.

[15] J. E. Beasley, "OR-Library: distributing test problems by electronic mail", *J. of the Oper. Res. Soc.*, vol. 41, no. 11, pp. 1069–1072, 1990.

[16] B. Abramson, "Expected-outcome: A general model of static evaluation", *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 2, pp. 182–193, 1990.

[17] R. Coulom, "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search", in *Computers and Games: 5th International Conference, CG 2006, Turin, Italy, May 29–31, 2006. Revised Papers*, H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers (Jeroen), Eds. Berlin Heidelberg: Springer, 2007, pp. 72–83.

[18] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem", *Machine Learning*, vol. 47, no. 2–3, pp. 235–256, 2002.

[19] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning", in *Machine Learning: ECML 2006. 17th European Conference on Machine Learning Berlin, Germany, September 18–22, 2006 Proceedings*, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds. Berlin Heidelberg: Springer, 2006, pp. 282–293, 2006.

[20] J. Derrac, S. Garcia, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms", *Swarm and Evol. Comput.*, vol. 1, no. 1, pp. 3–18, 2011.

[21] J. Alcala-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. Garcia, L. Sanchez, and F. Herrera, "KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework", *J. of Multiple-Valued Logic and Soft Comput.*, vol. 17, no. 2–3, pp. 255–287, 2011.

Michał Bereta received his B.Eng. degree in Information Technology from Jyväskylä Polytechnic (Finland) in 2003 and his M.Sc. degree in Computer Modeling from Cracow University of Technology in 2004. He received his Ph.D. degree in Computer Science from the Institute of Computer Science of the Polish Academy of Sciences in 2008. Between 2010–2012 he was a Postdoctoral Fellow at the Department of Electrical and Computer Engineering, University of Alberta. He works at the Institute of Computer Science of the Cracow University of Technology. His research interests include computational intelligence, machine learning, data mining and pattern recognition.

E-mail: mbereta@pk.edu.pl

Faculty of Physics, Mathematics and Computer Science
Tadeusz Kościuszko Cracow University of Technology
24 Warszawska st
31-155 Cracow, Poland