Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 31 (2014) 1074 – 1083

Procedia Computer Science

2nd International Conference on Information Technology and Quantitative Management, ITQM 2014

Heuristics for Minimum Spanning k-tree Problem

Roman E. Shangin[a], Panos M. Pardalos[b]

[a]76, Lenin prospekt, Chelyabinsk, Russia, 454080
[b]401 Weil Hall, P.O. Box 116595, Gainesville, FL 32611-6595

Abstract

In this paper we consider the problem of finding a spanning k-tree of minimum weight in a complete weighted graph which has a number of applications in designing reliable telecommunication networks. This problem is known to be NP-hard. We propose four effective heuristics: the first heuristic is based on the idea of a well-known Prim's algorithm, the second one is based on a dynamic programming approach, and the other two use the idea of iterative improvement from a starting solution. Preliminary numerical experiment was performed to compare the effectiveness of the proposed algorithms with known heuristics and exact algorithms.
© 2014 Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/3.0/).
Selection and peer-review under responsibility of the Organizing Committee of ITQM 2014.
Keywords: Spanning k-tree, invulnerable networks, NP-hard, heuristics;

1. Introduction

We consider a NP-hard combinatorial optimization problem of finding a spanning k-tree[1] of minimum weight in a complete weighted graph, known in literature as *Minimum Spanning k-Tree Problem* (MSkT). MSkT has a number of applications in designing reliable telecommunication networks and generalizes a classical problem in graphs, the *Minimum Spanning Tree Problem*[2].

Definition[1]**.** *A k-tree is a member of a class of undirected graphs defined recursively as follows: complete graph with k vertices is a k-tree; if T is a k-tree with n vertices, then a new k-tree with n + 1 vertices is formed by creating a new vertex v and adding edges between v and every vertex of an existing k-clique (clique with k vertices) in T.*

The mathematical formulation of the MSkT is as follows. Let $G = (V, E)$ be a complete weighted undirected graph, where V is a set of nodes and E is the set of edges, and for each edge $[i, j] \in E$ the weight $w(i, j) \geq 0$ is given. Let $T(G)$ be a set of all spanning k-trees in a graph G, where a spanning k-tree is a k-tree that contains all the vertices and a subset of the edges of a graph G. Let $w(T)$ be a weight of edges of the spanning k-tree $T \in T(G)$. It is required to find a spanning k-tree T^* of minimum weight in a complete weighted graph G: $T^* = \arg\min_{T \in T(G)}\{w(T)\}$.

* Corresponding author. Tel.: +7-912-306-0516.
E-mail address: shanginre@gmail.com

1877-0509 © 2014 Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/3.0/).
Selection and peer-review under responsibility of the Organizing Committee of ITQM 2014.
doi:10.1016/j.procs.2014.05.362

The first mention of the MSkT problem is given in Farley[3], in which he introduced the concept of isolated failure immune (IFI) networks, i.e. such networks remain connected even in the presence of a large number of failures. More specifically IFI networks work with three types of failures[3]:

i. $[i, j]$ and $[p, q]$ are two *line isolated failures* if $[i, j]$ and $[p, q]$ are not incident to a common node;
ii. i and j are two *node isolated failures* if i and j are not connected;
iii. A *line failure* $[i, j]$ and a *node failure* p are isolated if $[i, j]$ is not incident to p or to a node connected to p.

A set of failures is isolated if the failures in the set are pairwise isolated. The network is IFI if it remains connected as long as network failures are isolated. Farley proved that 2-trees are minimal (minimal respect to edge inclusion) IFI networks and proposed algorithms for finding a 2-tree of minimum cost in a weighted graph. After that, Bern[4] in his doctoral thesis proved that MSkT is NP-hard for $k \geq 2$ and designed a nonpolynomial exact algorithm based on a dynamic programming approach. Later, Cai and Maffray[5] strengthened his result by showing that the problem remains NP-hard for degree-bounded graphs, split graphs and planar graphs.

The great practical importance of MSkT problem and proof of its NP-hardness stimulated the development of heuristics and approximation algorithms for solving such a problem. Greedy heuristics for MS2T were proposed in Beck and Candia[6] and Candia et al.[7]. After that, Beltran and Skorin-Kapov[8] developed four heuristics to obtain good feasible solutions for MS2T. They also formulated an integer programming model which objective function value is a lower bound to the MS2T problem. In[9] Cai proposed approximating algorithms for MS2T. Later, H. Beck and A. Candia[10] proposed algorithms for MS2T that accomplish a greedy strategy based on the recursive definition of the spanning k-tree. Although MS2T is well studied, in literature little attention is given to the development of algorithms for solving the MSkT problem, where $k > 2$.

The main purpose of this work is to develop effective heuristics for solving MSkT problem (for $k \geq 2$) on a complete weighted graph. The plan of this paper is as follows. In section 2, we propose a heuristic that uses a greedy strategy and is based on the recursive definition of k-tree. In section 3, we propose a heuristic based on a dynamic programming approach. In section 4, we propose two effective algorithms, that use the idea of iterative improvement from a starting solution. In section 5 we present computational results and in section 6 we summarize our findings.

2. Heuristic based on greedy strategy

In this section, we propose the algorithm *GreedyA* (Greedy Algorithm), which computes feasible solutions for the MSkT. Algorithm *GreedyA* accomplishes a greedy strategy based on the recursive definition of a spanning k-tree and uses the idea of the well-known Prim's algorithm for Minimum Spanning Tree Problem.

Let $W_i = (V'_i, E'_i)$ be a complete graph with i vertices, i.e. $|V'_i| = i$. Let $T_i = (V_i, E_i)$ be a k-tree with i nodes and $K(T_i)$ is a set of all k-cliques in T_i.

ALGORITHM GreedyA

Input: A complete graph $G = (V, E)$ with positive edge lengths and an integer $k \geq 2$.
Output: A spanning k-tree T_{opt} in G.

Method:

Step 1. Find the minimum weight edge $[l, m]$ in G. Construct a graph W_2, where $V'_2 = \{l, m\}$ and $E'_2 = [l, m]$. Let $i = 2$.

Step 2. (Compute the starting clique of size $k + 1$ in graph G)
Find the vertex $m^* \in V \setminus V'_i$ for which the total weight $\sum_{j \in V'_i} w(m^*, j)$ of edges connecting this vertex m^* with the nodes of the complete graph W_i is minimal. Construct the complete graph W_{i+1} by including the computed vertex m^* and the corresponding edges $[m^*, j] : j \in V'_i$ in graph W_i. Let $i = i + 1$.
IF $|V'_i| = k + 1$, THEN $T_i = W_i$ and go to step 3, ELSE go to step 2.

Step 3. (Compute the k-tree T_{i+1} with $i + 1$ vertices)
Find the vertex $m^* \in V \setminus V_i$ and the clique $K^* \in K(T_i)$ for which the total weight $\sum_{j \in K^*} w(m^*, j)$ of edges connecting this vertex m^* with this clique K^* is minimal. Construct the k-tree T_{i+1} by including the computed vertex m^* and the corresponding edges $[m^*, j] : j \in K^*$ in k-tree T_i. Let $i = i + 1$.
IF $|V_i| = |V|$, THEN $T_{opt} = T_i$ and stop the algorithm, ELSE go to step 3.

Algorithm *GreedyA* is a modification of the known *GREEDY* heuristic proposed by Beck and Candia in[10] for solving MSkT problem. Basic difference of *GreedyA* from *GREEDY* is that it is a method of computing the starting clique of size $k+1$: in *GREEDY* it is chosen randomly and in *GreedyA* it is chosen according with procedures at the steps 1 and 2.

Proposition 1. *The computational complexity of the proposed algorithm GreedyA does not exceed* $O((|V|-k)^3 \cdot k)$.

Proof. At the step 1 for finding the cheapest edge in graph G one needs $O(|E|)$ operations. Step 2 can be implemented in $O((k-1)\cdot|V|)$ operations, because $O(|V|)$ operations are needed to compute new vertex $m^* \in V \setminus V'_i$ at each iteration of the step 2, and the quantity of such iterations at the step 2 is equal to $k-1$. At each iteration of the step 3 one needs $O((|V|-k)^2 \cdot k)$ operations, because there are $(|V|-k)\cdot k$ cliques of size k in k-tree T_i, and the number of choices of the new vertex $m^* \in V \setminus V_i$ at each iteration of the step 3 does not exceed $|V|-k$. As far as the number of iterations performed in step 3 is $|V|-k-1$, then at step 3 one requires $O((|V|-k)^3 \cdot k)$ operations. Based on this, the computational complexity of the algorithm *GreedyA* is $O((|V|-k)^3 \cdot k)$.

Figure 1 shows the ways of constructing a solution for the MSkT problem by the *GreedyA* heuristic.

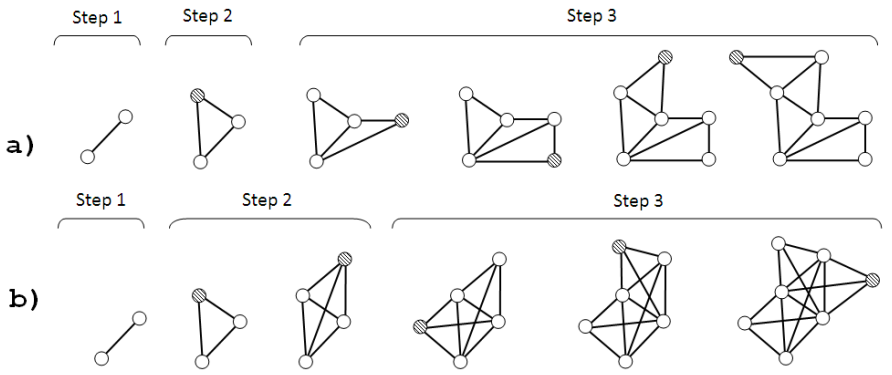

Fig. 1. Construction of the solutions by the *GreedyA* heuristic for: a) MS2T; b) MS3T.

3. Heuristic based on dynamic programming approach

In this section, we propose the algorithm *DPA* (Dynamic Programming Algorithm) based on dynamic programming (DP) approach for solving MSkT problem.

The process of solving the problem is divided into $|V|+1$ steps of the DP-process. Let us denote $V(i) : V(i) = V$ as a set of DP-process states during the step i. The value of the Bellman's function $f_i(j)$ which has been computed at the step i for a certain process state $j \in V(i)$ is the weight of the k-tree $T_i(j) = (V_i(j), E_i(j)) : T_i(j) \subset G$ with i vertices, which is constructed from k-tree $T_{i-1}(l)$ with $i-1$ vertices by including a new node j at the step i.

ALGORITHM DPA

Input: A complete graph $G = (V, E)$ with positive edge lengths and an integer $k \geq 2$.
Output: A spanning k-tree T_{opt} in G.

Method:

Step 1 (initial). For all states $j \in V(1)$ compute the graph $T_1(j) = (V_1(j), E_1(j))$ and the value of the Bellman's function $f_1(j)$ according to formulas:

$$V_1(j) = \{j\}; \quad E_1(j) = \emptyset; \quad f_1(j) = 0.$$

Step $i = 2, 3, ..., k+1$**.** For all $j \in V(i)$ and for any $l \in V(i-1)$ determine the set of the edges $\bar{E}_i(j,l)$ by adding the set $\{[j,m] : m \in V_{i-1}(l)\}$ of edges connecting the vertex j with vertices of the $T_{i-1}(l)$ to the set $E_{i-1}(l)$. For all $\bar{E}_i(j,l)$

compute the value $R_i(j,l)$ of the weight of such set of edges by summing the value $\sum_{m\in V_{i-1}(l)} w(j,m)$ of the total weight of edges connecting the vertex j with vertices of the $T_{i-1}(l)$ and the value $f_{i-1}(l)$ of the weight of the graph $T_{i-1}(l)$.

$$\bar{E}_i(j,l) = \left(\bigcup_{m\in V_{i-1}(l)} [j,m]\right) \cup E_{i-1}(l); \qquad R_i(j,l) = \sum_{m\in V_{i-1}(l)} w(j,m) + f_{i-1}(l).$$

Note that if $j \in V_{i-1}(l)$, then the value of the function $R_i(j,l)$ is $+\infty$.

For all DP-process states $j \in V(i)$ compute the graph $T_i(j) = (V_i(j), E_i(j))$ and the value of the Bellman's function $f_i(j)$ according to formulas:

$$V_i(j) = \{j\} \cup V_{i-1}(l^*); \qquad E_i(j) = \bar{E}_i(j,l^*) : l^* = \arg\min_{l\in V(i-1)} \{R_i(j,l)\}; \tag{1}$$

$$f_i(j) = \min_{l\in V(i-1)} \{R_i(j,l)\}. \tag{2}$$

Step $i = k+2, k+3, ..., |V|$**.** For all $j \in V(i)$ and for any $l \in V(i-1)$ determine the k-clique $K^*(j,l) \in K(T_{i-1}(l))$ for which the total weight $\sum_{m\in K^*(j,l)} w(j,m)$ of edges connecting the vertex j with this clique $K^*(j,l)$ is minimal. For all $j \in V(i)$ and for any $l \in V(i-1)$ compute the set of the edges $\bar{E}_i(j,l)$ by adding the set $\{[j,m] : m \in K^*(j,l)\}$ to the set $E_{i-1}(l)$. For all $\bar{E}_i(j,l)$ compute the value $R_i(j,l)$ of the weight of such set of edges.

$$\bar{E}_i(j,l) = \left(\bigcup_{m\in K^*(j,l)} [j,m]\right) \cup E_{i-1}(l); \qquad R_i(j,l) = \sum_{m\in K^*(j,l)} w(j,m) + f_{i-1}(l).$$

Note that, if $j \in V_{i-1}(l)$, then the value of the function $R_i(j,l)$ is $+\infty$.

For all DP-process states $j \in V(i)$ compute the graph $T_i(j) = (V_i(j), E_i(j))$ and the value of the Bellman's function $f_i(j)$ according to formulas (1) and (2), respectively.

Step $|V|+1$ **(final).** Compute the spanning k-tree $T_{opt} = (V, E_{opt})$ and the weight $w(T_{opt})$ of its edges according to formulas:

$$E_{opt} = E_{|V|}(j^*) : j^* = \arg\min_{j\in V}\{f_{|V|}(j)\}; \qquad w(T_{opt}) = \min_{j\in V}\{f_{|V|}(j)\}.$$

Stop.

Proposition 2. *The computational complexity of the proposed algorithm DPA does not exceed* $O(|V|^4 \cdot k)$.

Proof. Step 1 of the DP-process can be implemented in $O(|V|)$ operations, because $O(1)$ operations are needed to determine the sets $V_1(j)$, $E_1(j)$ and $O(1)$ operations are needed to compute the value of the Bellman's function $f_1(j)$ for all states $j \in V(1)$ of the DP-process, and the number of such process states is equal to $|V|$.

Step $i = 2, 3, ..., k+1$ of the DP-process can be implemented in $O(|V|^2)$ operations, because $O(|V|^2)$ operations are needed to compute the sets $\bar{E}_i(\cdot,\cdot)$, the values of the function $R_i(\cdot,\cdot)$ and the values of the Bellman's function $f_1(\cdot)$, and also $O(|V|)$ operations are required to determine the sets $V_i(\cdot)$ and $E_i(\cdot)$. Step $i = k+2, k+3, ..., |V|$ of the DP-process can be implemented in $O(|V|^3 \cdot k)$ operations, because $O(|V|^2 \cdot (|V|-k) \cdot k)$ operations are necessary to compute values of the function $R_i(\cdot,\cdot)$, $O(|V|^2)$ operations are needed to determine the sets $\bar{E}_i(\cdot,\cdot)$ and to compute the values of the Bellman's function $f_i(\cdot)$, and also $O(|V|)$ operations are needed to compute the sets $V_i(\cdot)$, $E_i(\cdot)$.

The finial step $|V|+1$ needs $O(|V|)$ operations, because $O(|V|)$ operations are required to compute the weight $w(T^*)$ and to determine the set of edges E^* of the spanning k-tree T^*. Based on this, computational complexity of the *DPA* is $O(|V|^4 \cdot k)$, where $O(|V|^4 \cdot k) = O(|V|) + O(|V|^2) \cdot k + O(|V|^3 \cdot k) \cdot (|V| - k - 1) + O(|V|)$.

4. Heuristics based on iterative improvement from a starting solution

We develop the heuristic *RA* (Recursive Algorithm) to obtain low-cost feasible solutions for the MSkT, based on the idea of iterative improvement from a starting solution computed by the proposed algorithms.

Idea of the algorithm *RA* is as follows. We compute the starting solution $T^* = (V, E^*)$ by the proposed algorithm *GreedyA* or *DPA*. After that, for all k-clique K in graph T^* construct the disconnected graph T_K by elimination all

edges from T^* which connect vertices of the clique K with the set of nodes $V \setminus K$. Next, the components of each graph T_K are connected by edges in order to minimize the total weight of the included edges, so that the resulting graph is a k-tree. If the weight of a constructed k-tree $T'_K = \arg\min_{K \in K(T^*)}\{w(T_K)\}$ is less than the weight of the initial graph T^*, then the graph T^* is replaced by a graph T'_K. The procedure is repeated as long as possible to improve the solution.

Before we present the recursive algorithm *RA*, it is better to give the procedure CONNECT that connects two cliques of size $\leq k$ by edges in order to minimize the total weight of the added ribs, so that the resulting subgraph satisfies property of a k-tree.

PROCEDURE CONNECT

Input: Cliques K_1 and K_2 of size $\leq k$, where $|K_1| \geq |K_2|$.
Output: Set of edges $\hat{E}(K_1, K_2)$ and the average weight of edges $\hat{w}(K_1, K_2)$ that connect the vertices of the cliques K_1 and K_2.

Method:

Let $V_2^* = \emptyset$ and $\hat{w}(K_1, K_2) = 0$.

Step 1. Find the vertex $m^* \in K_2 \setminus V_2^*$ for which the total weight $\sum_{j \in K_1} w(m^*, j)$ of edges connecting this vertex m^* with the clique K_1 is minimal. Include the set of edges $\{[m^*, j] : j \in K_1\}$ in $\hat{E}(K_1, K_2)$. Let $\hat{w}(K_1, K_2) = \hat{w}(K_1, K_2) + \sum_{j \in K_1} w(m^*, j)$ and $V_2^* = V_2^* \cup \{m^*\}$.
IF $|V_2^* \cup K_1| = k + 1$, THEN $V_1^* = K_1$ and go to step 2, ELSE go to step 1.

Step 2. Compute the set $\bar{V}_1 = \{V_1^* \setminus \{j\} : j \in V_1^*\}$ of all cliques of size $|V_1^*| - 1$ whose vertices belong to V_1^*. It is clear that $|\bar{V}_1| \leq k$.
Find the vertex $m^* \in K_2 \setminus V_2^*$ and the clique $K^* \in \bar{V}_1$ for which the total weight $\sum_{j \in K^*} w(m^*, j)$ of edges connecting this vertex m^* with this clique K^* is minimal. Include the set of edges $\{[m^*, j] : j \in K^*\}$ in $\hat{E}(K_1, K_2)$ and compute $\hat{w}(K_1, K_2) = \hat{w}(K_1, K_2) + \sum_{j \in K^*} w(m^*, j)$. Let $V_2^* = V_2^* \cup \{m^*\}$ and $V_1^* = V_1^* \cap K^*$.
IF $V_2^* = K_2$, THEN

$$\hat{w}(K_1, K_2) = \frac{\hat{w}(K_1, K_2)}{|\hat{E}(K_1, K_2)|}$$

and stop the procedure, ELSE go to step 2.

Proposition 3. *The computational complexity of the procedure CONNECT does not exceed* $O(k^3)$.

Proof. For computing the vertex $m^* \in K_2$ in step 1 we need not more than $O(k)$ operations, and the number of such vertices included at step 1 into set V_2^* is at least equal 1. Naturally, for computing the vertex $m^* \in K_2$ at step 2 one requires $O(|K_2 \setminus V_2^*| \cdot |\bar{V}_1|)$ operations. In accordance with this, the total number of operations does not exceed $k + \sum_{i=1}^{k-1}(k-i)(k-i+1)$, where

$$k + \sum_{i=1}^{k-1}(k-i)(k-i+1) = k + (k-1)k + (k-2)(k-1) + (k-3)(k-2) + ... + 0 =$$

$$k + k^2(k-1) - \frac{k^2(k-1)}{2} - \frac{k^2(k-1)}{2} + \frac{k(k-1)(k+1)}{3} = k + \frac{k(k-1)(k+1)}{3}.$$

Based on this, the computational complexity of the procedure CONNECT is $O(k^3)$.

We also give a procedure COMPLEMENT, which builds a k-tree from a partial k-tree. By a partial k-tree we shall further mean a graph that contains all the vertices and a subset of the edges of a k-tree. The idea of the procedure consists in the consecutive inclusion of edges into a partial k-tree in order to minimize weight of included edges so that the graph which is constructed by adding edges is a k-tree.

PROCEDURE COMPLEMENT

Input: A partial k-tree $T_p = (V_p, E_p)$.
Output: A k-tree $T' = (V_p, E')$.

Method:

Let $T' = T_p$ and $E_G = \{[i, j] : i, j \in V_p\} \setminus E_p$. Sort E_G in increasing order of weights.
WHILE $|E'| < k(2|V_p| - k - 1)/2$:
Let $[x, y]$ be the minimum weight edge in E_G.
IF $(V_p, E' \cup [x, y])$ is a partial k-tree, THEN $E' = E' \cup [x, y]$.
Let $E_G = E_G \setminus [x, y]$.
ENDWHILE

Stop.

The computational complexity of the procedure COMPLEMENT is $O(|V_p|^3 \cdot k)$, because the number of edges in a k-tree $T' = (V_p, E')$ is equal to $k(2|V_p| - k - 1)/2$ and $O(|V_p|^2)$ operations are required for answering to a question whether a graph is a partial k-tree or not [11,12].

ALGORITHM RA

Input: A complete graph $G = (V, E)$ with positive edge lengths and an integer $k \geq 2$. A starting solution $T^* = (V, E^*)$, computed by some heuristic algorithm.
Output: A spanning k-tree T_{opt} in G.

Method:

Iteration $i : i = 1, 2, ..., I$

Stage 1. For every clique $K \in K(T^*)$ of size k construct the graph T_K by removal edges from k-tree T^* which connect vertices of the k-clique K with nodes of the graph T^* which do not belong to K.

Stage 2. For all graphs $T_K : K \in K(T^*)$ perform steps 1-5:
Step 1. From every connected component of size $> k$ build the k-tree with the help of the procedure COMPLEMENT, and if the size of a some connected component is $< k$, then remove all edges from such a component.
Step 2. For all connected components $C = (V_C, E_C)$ of the graph T_K define the set $K(C)$ of all k-cliques, and if the number of vertices of some component is $\leq k$, then $K(C) = V_C$.
Step 3. For all pair of cliques $K_1 \in K(C_1)$ and $K_2 \in K(C_2)$ that belong to different connected components, i.e. $C_1 \neq C_2$, determine the set $\hat{E}(K_1, K_2)$ and the weight $\hat{w}(K_1, K_2)$ of edges, with the help of the procedure CONNECT.
Step 4. Include in graph T_K the set of edges $\hat{E}(K_1^*, K_2^*)$ that corresponds to the smallest value of the $\hat{w}(K_1, K_2)$ computed for a cliques belonging to different connected components. IF the number of components of T_K is more than 1, THEN go to step 5 of the stage 2, ELSE if all disconnected graphs $T_K : K \in K(T^*)$ constructed into k-trees – go to stage 3.
Step 5. Define the set $K(C')$ of all k-cliques of the component C', obtained by connecting components $C_1 : K_1^* \in V_{C_1}$ and $C_2 : K_2^* \in V_{C_2}$ at the step 4, and if $|V_{C'}| \leq k$, then $K(C') = V_{C'}$.
Step 6. For all pair of cliques $K' \in K(C') : K' \notin K(C_1) \cup K(C_2)$ and $K \in K(C)$ that belong to different components determine the set $\hat{E}(K', K)$ and the weight $\hat{w}(K', K)$ of edges, with the help of the procedure CONNECT. Go to step 4 of the stage 2.

Stage 3. For all k-cliques $K \in K(T^*)$ compute the value $w(T_K)$ of the weight of the k-tree T_K, which has been constructed at the stage 2.
IF $w(T^*) > \min_{K \in K(T^*)}\{w(T_K)\}$, THEN $T^* = \arg\min_{K \in K(T^*)}\{w(T_K)\}$ and go to stage 1 of next iteration $i + 1$, ELSE $T_{opt} = T^*$ and stop the algorithm.

Proposition 4. *The computational complexity of the proposed algorithm RA does not exceed* $O((|V| - k)^4 \cdot k^3 \cdot I)$.

Proof. Stage 1 can be implemented in $O((|V| - k) \cdot k)$ operations, because number of k-cliques in the k-tree T^* equal to $(|V| - k) \cdot k$. At the step 1 of the stage 2 one requires $O(|V|^4 \cdot k^2)$ operations, because $O(|V|^3 \cdot k)$ operations are required for building the k-tree from every connected component of the graph T_K, and the number of graphs T_K which have been constructed at the stage 1 equal to $(|V| - k) \cdot k$. At the step 2 of the stage 2 one requires $O((|V| - k)^2 \cdot k^2)$ operations, because the number of k-cliques in the k-tree T^* is equal to $(|V| - k) \cdot k$, and the number of such graphs that have been constructed at the stage 1 is equal to $(|V| - k) \cdot k$. At the step 3 of the stage 2 one requires $O((|V| - k)^3 \cdot k^6)$ operations, because the number of pairs of cliques $K_1 \in K(C_1)$ and $K_2 \in K(C_2)$ that belong to different component of the graph T_K does not exceed $O((|V| - k)^2 \cdot k^2)$, and $O(k^3)$ operations are required for determining a set $\hat{E}(K_1, K_2)$ and the value $\hat{w}(K_1, K_2)$ for each pair of cliques K_1 K_2, and the number of graphs T_K which have been constructed at the

stage 1 equal to $(|V| - k) \cdot k$. At the step 4 of the stage 2 one requires $O((|V| - k)^4 \cdot k^3)$ operations, because the number of values $\hat{w}(K_1, K_2)$ that is computed at the step 2 does not exceed $O((|V| - k)^2 \cdot k^2)$, and the number of connected components of a graph T_K does not exceed $|V| - k$, and the number of such graphs that have been built at the stage 1 is equal to $(|V| - k) \cdot k$. At the step 5 of the stage 2 one requires $O((|V| - k)^3 \cdot k^2)$ operations, because the number of k-cliques in the component C' does not exceed $(|V| - k) \cdot k$, and the number of connected components of a graph T_K does not exceed $|V| - k$, and the number of such graphs that have been built at the stage 1 is equal to $(|V| - k) \cdot k$. At the step 6 of the stage 2 one requires $O((|V| - k)^3 \cdot k^7)$ operations, because the number of new cliques that are formed by connecting of two components does not exceed $O(k^2)$, $O(k^3)$ operations are required for determining a set $\hat{E}(K_1, K_2)$ and the value $\hat{w}(K_1, K_2)$, the number of k-cliques in the T_K does not exceed $(|V| - k) \cdot k$, and the number of connected components of a graph T_K does not exceed $|V| - k$, and the number of such graphs that have been built at the stage 1 is equal to $(|V| - k) \cdot k$. Stage 3 can be implemented in $O((|V| - k) \cdot k)$ operations, because the number of connected k-trees T_K that have been built at the stage 2 is equal to $(|V| - k) \cdot k$. Based on this, computational complexity of the *RA* is $O((|V| - k)^4 \cdot k^3 \cdot I)$, where I is a number of iterations.

Figure 3 shows the constructing the graph T_K from the graph T^* by the *RA* heuristic.

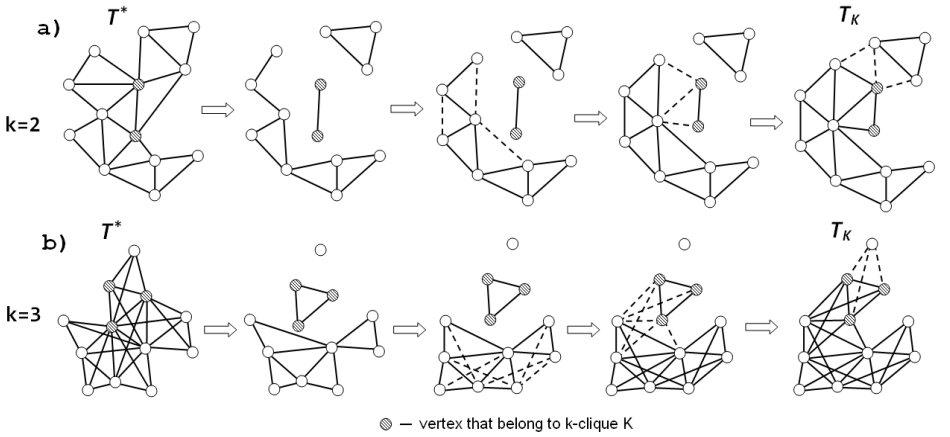

Fig. 2. The constructing the graph T_K from the graph T^* for: a) MS2T; b) MS3T.

Obviously, using of the proposed iterative algorithm for solving a MSkT problem of large and extra large dimension is difficult, due to the high computational complexity of the algorithm. On this basis, we propose the following iterative algorithm *FRA* (Fast Recursive Algorithm), which partially uses the idea of the *RA* algorithm, but having much lower computational complexity.

Let $E_T(K)$ be a set of edges of k-tree T that connect vertices of a clique K with other vertices of the graph T. Let $\bar{w}(K)$ be a value of average weights of the edges that connect vertices of clique K of size k with other vertices of k-tree T:

$$\bar{w}(K) = \frac{\sum_{[m,j] \in E_T(K)} w(m, j)}{|E_T(K)|}.$$

ALGORITHM FRA

Input: A complete graph $G = (V, E)$ with positive edge lengths and an integer $k \geq 2$. A starting solution T^*, computed by some heuristic. A positive integer $n : n \leq |V|$.
Output: A spanning k-tree T_{opt} in G.

Method:

Iteration $i : i = 1, 2, ..., I$

Stage 1. For all cliques K of size k in the graph T^* compute the value $\bar{w}(K)$. Sort k-cliques of the graph T^* in increasing order of the value $\bar{w}(K)$ and choose n first cliques, that it's set of vertices does not intersect. In the k-tree

T^* delete edges that connect vertices of each selected clique with other nodes of the graph T^*.

Stage 2. Build a k-tree T' from the disconnected graph T^* by performing steps 1-6 at the stage 2 of the algorithm *RA*.

Stage 3. IF $w(T^*) > w(T')$, THEN $T^* = T'$ and go to stage 1, ELSE let $T_{opt} = T^*$ and stop the algorithm.

Stop.

Obviously, the computational complexity of the algorithm *FRA* does not exceed $O((|V| - k)^3 \cdot k^2 \cdot I)$, since this algorithm considers a single graph obtained by removing edges from the k-tree T^* and the *RA* algorithm considers $(|V| - k) \cdot k$ such graphs.

5. Experimental results

Preliminary numerical experiment was performed to compare the effectiveness of the proposed algorithms with known heuristics and exact algorithms. By efficiency we understand the running time and the solution accuracy. We used sets of graphs, and each set included 15 complete graphs of the same dimension, with edge weights randomly generated. Calculations have been performed on a PC with a processor Intel Core i5 2.6 GHz.

Table 1 shows the results of numerical experiment based on the analysis of the effectiveness of the proposed algorithms compared with known algorithms for MS2T problem: *GREEDY* heuristic and its repetitive version *GREEDY1*, which obtains $|V|$ solutions, such that at each repetition, step 1 chooses a different starting k-clique within the first $|V|$ k-cliques of minimum length (proposed by Beck and Candia in [10]); *STARS* heuristic, which is based on an embedding of a minimum cost spanning tree into a 2-tree, *GREEDY PARTIAL* heuristic, which in each step adds the minimum weight available edge that does not induce a K_4-homeomorphic subgraph and *TABU SEARCH* heuristic, which starts with a feasible solution obtained by *GREEDY PARTIAL* heuristic and its search strategy alternates between edge based and recursion based moves (*STARS*, *GREEDY PARTIAL* and *TABU SEARCH* proposed by Beltran and Skorin-Kapov in [8]); BernA, which finds an exact solution of the MSkT problem (proposed by Bern in [4]).

We denote $\bar{t}_{alg}$ – average running time of an algorithm, sec.; $\bar{\varepsilon}_{alg}$ – average relative error of the algorithm, %: $\varepsilon_{alg} = \frac{w_{alg} - w_{opt}}{w_{opt}}$, where w_{alg} – weight of a spanning k-tree constructed by the algorithm, and w_{opt} – weight of a spanning k-tree constructed by exact algorithm BernA. We shall further denote, for example *FRA+DPA* – heuristic *FRA* with starting solution computed by algorithm *DPA*.

Table 1. Experimental results for MS2T of dimension $|V| \leq 12$.

Algorithms		$\|V\| = 5$	$\|V\| = 8$	$\|V\| = 10$	$\|V\| = 12$
GreedyA	$\bar{\varepsilon}_{alg}$	0,0805	0,4572	0,6586	0,8791
DPA	$\bar{\varepsilon}_{alg}$	0,0437	0,1737	0,3845	0,7794
RA+GreedyA	$\bar{\varepsilon}_{alg}$	0	0	0	0,0031
RA+DPA	$\bar{\varepsilon}_{alg}$	0	0	0	0
FRA+GreedyA	$\bar{\varepsilon}_{alg}$	0	0,0163	0,0971	0,2391
FRA+DPA	$\bar{\varepsilon}_{alg}$	0	0,0921	0,0528	0,1924
GREEDY	$\bar{\varepsilon}_{alg}$	0,0633	0,5309	0,7948	1,2937
GREEDY1	$\bar{\varepsilon}_{alg}$	0,0546	0,3545	0,4735	0,8067
STARS	$\bar{\varepsilon}_{alg}$	0,1122	1,8211	2,9067	3,5512
GREEDY PARTIAL	$\bar{\varepsilon}_{alg}$	0,0523	0,3924	0,5852	0,8129
TABU SEARCH	$\bar{\varepsilon}_{alg}$	0,0223	0,1265	0,3218	0,7279
BernA	$\bar{t}_{alg}$	0,0337	3,4116	62,6632	907,1102

Based on the results presented in Table 1 it follows that the accuracy of the solution proposed by algorithm *GreedyA* is better than *STARS* heuristic, but worse than heuristics *GREEDY1*, *GREEDY PARTIAL* and *TABU SEARCH*. Proposed heuristic *DPA*, based on dynamic programming, surpassed all presented in Table 1 known algorithms in terms of solution accuracy, except heuristic *TABU SEARCH*. Heuristic *RA+DPA* showed the best results in terms of accuracy solutions in comparison with all the algorithms presented in Table 1. In addition, for all generated tasks it has computed the optimal solution.

Obviously, the exponential increase of the computation time of the exact algorithm BernA makes it unsuitable for solving the MS2T problem of medium and large dimension. Based on this, Table 2 shows the results of the numerical experiment on the analysis of the effectiveness of the proposed algorithms for solving MS2T, where the accuracy of

Table 2. Experimental results for MS2T of dimension $|V| \geq 15$.

Algorithms		$\|V\| = 15$	$\|V\| = 25$	$\|V\| = 50$	$\|V\| = 75$	$\|V\| = 100$
	$\bar{t}_{alg}$	0,0164	0,0368	0,2814	0,7931	1,9334
GreedyA	$\bar{\varepsilon}_{alg}$	1,7138	4,3234	6,0394	8,362	8,4709
	$\bar{t}_{alg}$	0,0741	0,5349	8,8329	48,0729	137,3226
DPA	$\bar{\varepsilon}_{alg}$	1,2445	3,5238	5,3629	5,8567	7,0394
	$\bar{t}_{alg}$	0,1159	1,0978	22,7021	123,4843	413,6907
RA+GreedyA	$\bar{\varepsilon}_{alg}$	0,0129	0,0352	0,0438	0,0594	0,0699
	$\bar{t}_{alg}$	0,0178	0,0788	0,6144	2,0292	5,2218
FRA+GreedyA	$\bar{\varepsilon}_{alg}$	0,4799	1,2352	1,6283	2,1628	2,3837
	$\bar{t}_{alg}$	0,0776	0,6399	8,9753	43,7251	154,5231
FRA+DPA	$\bar{\varepsilon}_{alg}$	0,4037	1,0397	1,4865	2,0059	1,7984
	$\bar{t}_{alg}$	0,0108	0,0348	0,2782	0,7064	1,8584
GREEDY	$\bar{\varepsilon}_{alg}$	1,7922	5,0456	6,8996	7,9616	9,5122
	$\bar{t}_{alg}$	0,1031	0,829	12,8532	58,4499	211,0878
GREEDY1	$\bar{\varepsilon}_{alg}$	1,4511	3,3928	5,7852	7,3256	6,9879
	$\bar{t}_{alg}$	0,0112	0,039	0,2682	0,9358	2,4492
STARS	$\bar{\varepsilon}_{alg}$	4,5304	13,0916	16,6422	22,1154	22,6415
	$\bar{t}_{alg}$	0,0071	0,0206	0,1359	0,3976	1,0744
GREEDY PARTIAL	$\bar{\varepsilon}_{alg}$	1,3723	3,7225	5,0536	6,8829	8,3098
	$\bar{t}_{alg}$	0,7151	1,6568	6,4314	12,9789	23,4419
TABU SEARCH	$\bar{\varepsilon}_{alg}$	1,0898	2,9584	4,1018	5,0346	5,0684
RA+DPA	$\bar{t}_{alg}$	0,2065	1,7463	27,9932	158,5894	566,4117

Table 3. Experimental results for MSkT of dimension $|V| = 50$.

Algorithms		$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$	$k = 8$
	$\bar{t}_{alg}$	0,3708	0,4854	0,6306	0,8139	0,8582	0,9808
GreedyA	$\bar{\varepsilon}_{alg}$	7,4855	9,4197	11,0749	10,7975	14,3207	16,7264
	$\bar{t}_{alg}$	12,1429	16,6365	23,1116	28,3245	34,1535	39,0326
DPA	$\bar{\varepsilon}_{alg}$	6,8653	7,8187	7,5329	9,1135	10,4064	11,2684
	$\bar{t}_{alg}$	48,4445	46,4435	41,5792	44,8639	49,8492	49,9883
RA+GreedyA	$\bar{\varepsilon}_{alg}$	0,0504	0,0618	0,0638	0,0731	0,0826	0,0879
	$\bar{t}_{alg}$	0,7996	1,0543	1,0789	1,1705	1,4521	1,5873
FRA+GreedyA	$\bar{\varepsilon}_{alg}$	2,2451	2,452	2,5869	3,0163	2,685	2,8807
	$\bar{t}_{alg}$	13,6688	19,4276	24,5049	25,4076	28,8294	32,886
FRA+DPA	$\bar{\varepsilon}_{alg}$	1,7981	2,1468	2,1757	2,1599	2,5622	2,7489
	$\bar{t}_{alg}$	0,3672	0,5286	0,6425	0,6927	0,9235	1,0554
GREEDY	$\bar{\varepsilon}_{alg}$	9,0908	11,1129	16,856	21,3063	24,8551	28,5144
RA+DPA	$\bar{t}_{alg}$	58,9924	67,8281	69,0328	70,6682	80,6341	85,3666

the solution of algorithms was estimated in comparison with heuristic *RA+DPA*, i.e. in Table 2 value ε_{alg} of relative error of the algorithm is calculated by the formula $\varepsilon_{alg} = \frac{w_{alg} - w_{RA}}{w_{RA}}$, where w_{alg} – weight of spanning k-tree computed by algorithm, and w_{RA} – weight of spanning k-tree computed by proposed algorithm *RA+DPA*.

Based on the results presented in Table 2 it follows that *DPA* heuristic surpassed all presented in Table 1 known algorithms in terms of the solution accuracy, except *TABU SEARCH* heuristic. Algorithm *RA* that uses an initial solution computed by *GreedyA* heuristic had the smallest error for all series of tasks. The algorithm *FRA+GreedyA* in terms of all performance criteria surpassed known heuristic *TABU SEARCH*. Note that heuristic *TABU SEARCH* obtains a better solution for only 1.2% of the generated tasks in comparison with the algorithm *RA+DPA*.

Table 3 shows results of computational experiments to analyze the effectiveness of the proposed heuristics and known *GREEDY* algorithm for solving the MSkT problem of dimension $|V| = 50$ for $k = 3, 4, ..., 8$. Note that in Table 3, the value ε_{alg} of relative error of an algorithm is calculated the same way as in Table 2.

Based on the results presented in Table 3 it follows that *GREEDY* algorithm showed the maximum growth of error by increasing value k, such as increase of the average error of solving the MSkT problem for $k = 8$ with respect to error of solving the problem for $k = 3$ was 215%, when an increase of the error of the proposed algorithm *GreedyA* did not exceed 123% and algorithm *DPA* 64%. Heuristic *FRA+GreedyA* showed the smallest increase of error by increasing values k, for example increase of the average error of solving the MSkT problem for $k = 8$ with respect to

error of solving the problem for $k = 3$ was less than 30%, besides with increasing values k, the difference between the error of the algorithm *FRA+GreedyA* and the error of the algorithm *FRA+DPA* decreases.

Conclusions

The paper considers the well-known NP-hard problem of finding a spanning k-tree of minimum weight in a complete weighted graph. We proposed four effective heuristic algorithms, the first algorithm *GreedyA* uses the idea of the known Prim's algorithm, the second algorithm *DPA* is based on a dynamic programming approach, and the other two use the idea of iterative improvement for starting solution that is computed by algorithms *GreedyA* or *DPA*. Preliminary numerical experiment was performed to compare the effectiveness of the proposed algorithms with known heuristics and exact algorithms.

Based on the results of the computational experiment for the problem MS2T it follows that the accuracy of a solution of the proposed algorithm *GreedyA* is better than *STARS* heuristic, but worse than *GREEDY1*, *GREEDY PARTIAL* and *TABU SEARCH* heuristic. Proposed heuristic *DPA*, based on dynamic programming, surpassed all presented in Table 1 known algorithms in terms of the solution accuracy, except *TABU SEARCH* heuristic. Heuristic *RA+DPA* showed the best results in terms of the accuracy of solutions in comparison with all algorithms presented in Table 1, in addition, for all generated tasks it computes the optimal solution. The algorithm *RA* that uses an initial solution computed by heuristic *GreedyA* received a least error for all series of tasks. Algorithm *FRA+GreedyA* in terms of all performance criteria surpassed *TABU SEARCH* heuristic. Note that *TABU SEARCH* heuristic for only 1.2% of the generated tasks gets a better solution than the algorithm *RA+DPA*.

Based on the results of the computational experiment for the problem MSkT it follows that *GREEDY* algorithm showed the maximum growth of the error by increasing the value k, such as increase of the average error of solving the MSkT problem for $k = 8$ with respect to the error of solving the problem for $k = 3$ was 215%, when an increase of the error of the proposed algorithm *GreedyA* did not exceed 123% and algorithm *DPA* 64%. Heuristic *FRA+GreedyA* showed the smallest increase of the error by increasing values k, for example increase of the average error of solving the MSkT problem for $k = 8$ with respect to the error of solving the problem for $k = 3$ was less than 30%, besides, with increasing values k, the difference between the error of the algorithm *FRA+GreedyA* and the error of the algorithm *FRA+DPA* decreases.

On this basis, it follows that in order to solve MSkT of small and medium dimension it is advisable to use *RA+DPA* and *RA+GreedyA* heuristic, and in order to solve the MSkT problem of high dimension it is advisable to use an algorithm *FRA+GreedyA*, because it computes a solution with sufficient accuracy within reasonable computing time.

Acknowledgements

The authors are grateful to corresponding member of Russian Academy of Sciences, professor A. G. Chentcov for constructive criticism and interest in this problem.

References

1. Rose D. On simple characterizations of k-trees. *Discrete Mathematics* 1974;**41**:317-322.
2. Prim R. Shortest connection networks and some generalizations. *Bell Systems Techn. J.* 1957;**36**:1389-1401.
3. Farley A. Networks immune to isolated failures. *Networks* 1981;**11**:255-268.
4. Bern M. *Networks Design Problems: Steiner Trees and Spanning k-Trees, Ph. D. Thesis*. University of Berkeley; 1987.
5. Cai L., Maffray F. *On the spanning k-tree problem*. University of Toronto; 1992.
6. Beck H., Candia A. An heuristic for the minimum spanning 2-tree problem. *Apuntes de Ingeniera (Special number in Computer Science)* 1993;**47**:97-109.
7. Beck H., Candia A., Bravo H. Optimal design of invulnerable networks. *Research Report* 1993;**15**:107-113.
8. Beltran H., Skorin-Kapov D. On minimum cost isolated failure immune network. *Telecommunications System Modeling and Analysis* 1993;**12**:444-453.
9. Cai L. *On spanning 2-trees in a graph*. University of Hong Kong; 1996.
10. Beck H., Candia A. Heuristics for minimum spanning k-trees. *Investigation Operativa* 2000;**9**:104-116.
11. Wald J., Colbourn C. Steiner trees, partial 2-trees and minimum IFI networks. *Networks* 1983;**13**:159-167.
12. Bodlaender H. Improved self-reduction algorithms for graphs with bounded treewidth. *Discrete Applied Mathematics* 1994;**54**:101–115.