Dynamic Programming Usage in Steiner Problem in Graphs

Joseph V. Romanovsky Dmitry A. Eibozhenko

Third Russian Finnish Symposium on Discrete Mathematics,
September 2014

Steiner tree problem statement

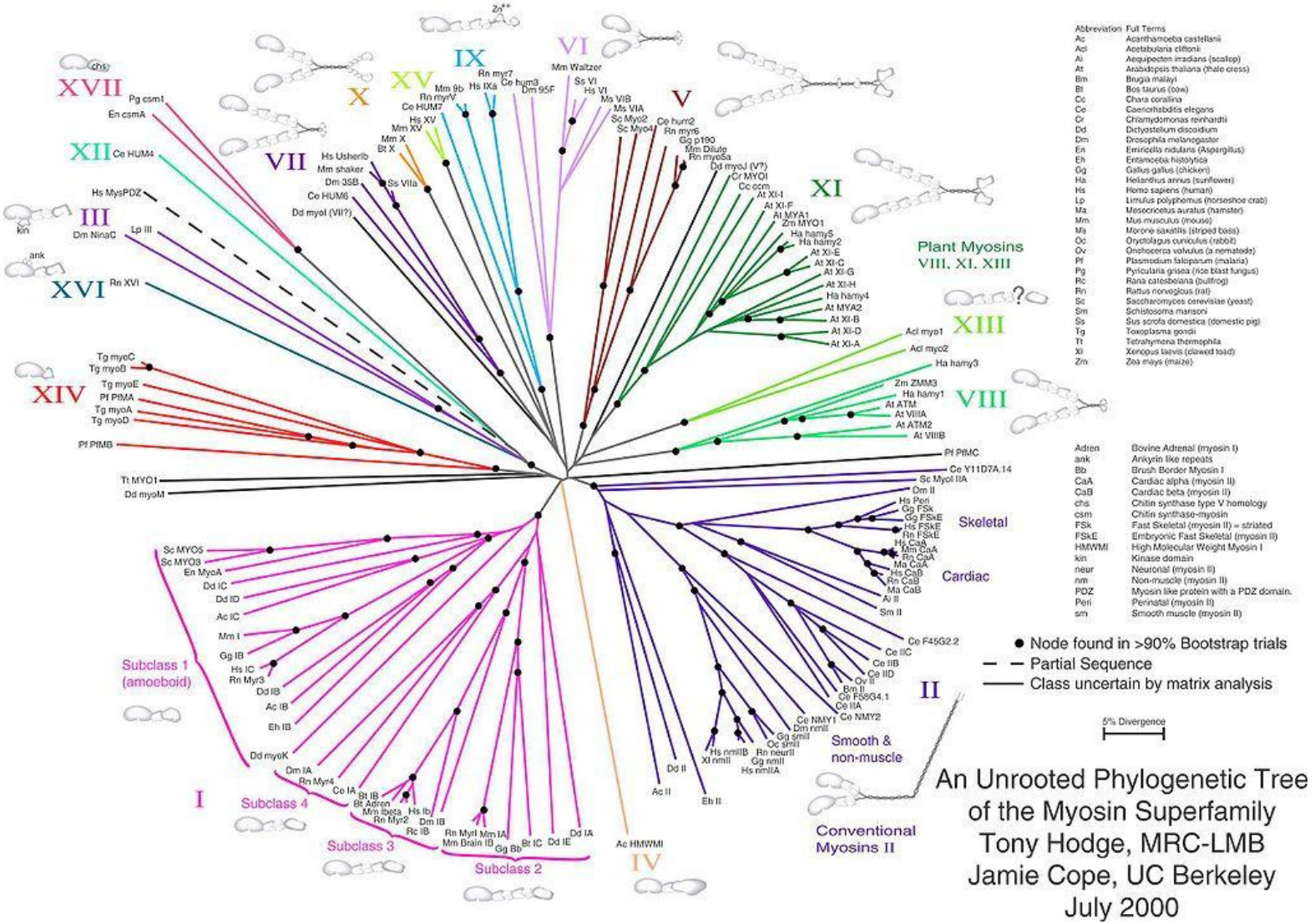
Steiner tree in directed graphs (arborescence). Let $G(M, N)$ be a connected directed graph (*digraph*) with cost function $d : N \to \mathbb{R}_+$. Begin node b and set of terminal nodes E are marked out in M. The problem is to find tree of minimal cost rooted in begin node b, and containing paths from b to any node of E.
Other statements:

- euclidean (geometric);
- rectilinear;
- in networks.

Application

- **VLSI-design**: *Very Large Scale Integration*. Process of building integrated circuits combining millions transistors. Well-known central processing units (CPU) and graphics processing units (GPU) are the examples of VLSI-devices.
- **Telecommunications**: New formats (interactive TV, interactive video conference), new services (*Video-On-Demand*, ability to choose translation quality).
- **Reconstruction of phylogenies**: Phylogenetic tree represents evolutional interconnections between various species, having common ancestor. In graph $G(M, N)$ M is set of nucleotide sequences, $N = M \times M$ (complete graph), and for each edge its length is Hamming distance between appropriate chains.

Phylogenetic tree example



Implementation Challenge

In December, 2014 will take place 11th DIMACS Implementation Challenge in Collaboration with ICERM: Steiner Tree Problems.

More than 10 statements of Steiner problem: classical Steiner problem in graphs; classical geometric Steiner problem; prize-collecting Steiner trees; generalized Steiner trees (Steiner forest); obstacle-avoiding Steiner trees, etc

Organizers and advisory – most famous contributors:

- Renato F. Werneck, Microsoft Research, USA
- Martin Zachariasen, University of Copenhagen, Denmark
- Alexander Zelikovsky, Georgia State University, USA and others.

Takahashi-Matsuyama algorithm

Presented at 1980. Greedy algorithm.

At any time of costructing process there is subgraph $T_S = (M^*, N^*)$,

Initial state: $M^* = \{b\}$, $N^* = \varnothing$.

Step: Find $e^* = \operatorname{argmin}\{d(M^*, e) | e \in E \setminus M^*\}$. Then $T_S := T_S \cup P[M^*, e^*]$, where $P[M^*, e^*]$ — minimal among all possible ways from nodes M^* to e^*.

- Computational complexity: $O(tn^2)$ ($n = |N|$, a $t = |E|$)
- Precision: $2(1 - 1/t)$

General scheme of greedy contraction

Let G_E be a subgraph of G, induced by $E \cup \{b\}$.
$Mst(E)$ is minimal spanning tree. $M_0(E) = c(Mst(E))$.

Contraction of tree T: Lengths of arcs in $Mst(E)$ with ends in $E(T) \to 0$.

Init: $L = \varnothing$ – list of trees
while $M_0(E) > 0$
    Find full Steiner tree T^* in some class K,
    that minimize estimating function $f(T)$:
    $T^* \leftarrow \operatorname{argmin}_{T \in K} f(T)$.
    insert T^* in list L : $L = L \cup \{T^*\}$
    contract T^*
end while
Re-establish Steiner tree from trees in list L and return it.

Methods that use this scheme

- **Takahashi and Matsuyama algorithm**: K consists of all available paths, and $f(T) = c(T)$.
- **Reyward-Smith heuristic algorithm** : K is composed of all kinds of stars, and $f(T) = \frac{c(T)}{r-1}$, where r — count of leafs of T.
- **Generalized greedy heuristic**: K is composed of trees with three terminals and $f(T) = c(T) - (M_0(E) - M_0(E \setminus T))$.
- **Relative greedy heuristic with size constraints**: K is composed of all trees with no more than r terminals, $f(T) = \frac{c(T)}{M_0(S) - M_0(S \setminus T)}$.

Using LP & ILP

There are various formal representations.

$\rho = \{W \subseteq M \setminus \{b\}, W \bigcap E \neq 0\}$ is *directed cut*
$\delta^-(W) = \{[b, e] | e \in W, b \notin W\}$

$$\min \quad \sum_{n \in N} c_n x_n, \tag{1}$$

$$\sum_{n \in \delta^-(W)} x_n \geq 1, \quad \forall W \in \rho, \tag{2}$$

$$x_n \in \{0, 1\}, \quad \forall n \in N, \tag{3}$$

(3) $\rightarrow$: $x_n \geq 0, \ \forall n \in N$.
Proven integrality gap for relaxation is 2.

Dynamic programming algorithm

Based on Bellman equation.
Solving simultaneously $\forall i \in M, \forall E_p \subset E$.
$v(i, E_p)$ — solution of corresponding subtask.

$$v(i, E_P) = \min\{v_{\text{cont}}(i, E_P), v_{\text{part}}(i, E_P)\} \tag{4}$$

- v_{cont} — minimal cost of going by some arc

$$v_{\text{cont}}(i, E_P) = \min\{c_j + v(\textbf{end}\, j, E_P) | \textbf{beg}\, j = i\}, \tag{5}$$

- v_{part} — minimal cost of splitting terminal set E_P into two disjunctive subsets A и $E_P \setminus A$

$$v_{\text{part}}(i, E_P) = \min\{v(i, A) + v(i, E_P \setminus A) | A \subset E_P\}. \tag{6}$$

DP algorithm. Implementation.

Table of size $|M| \times 2^{|E|}$ – **L**.
L[i,j] – $v(j, E_i)$ at calculation time.
(any subset numeration, that satisfy: $S_i \subset S_k \Rightarrow i < k$)

At i-th iteration:

1. **Dijkstrification:** finding by modified Dijkstra method tree lengths for S_i.
2. **Correction:** $\forall i_0 < i$, such that $E_{i_0} \cap E_i = \varnothing$ examine $E_{i_1} = E_i \cup E_{i_0}$:

   $$L[i_1, j] = \min(L[i_1, j], L[i, j] + L[i_0, j])$$

k-Cluster Algorithm

Core idea

Solve Steiner trees for restricted number of terminals.

k is upper bound for number of terminals in tree, that can be solved by DP algorithm.

If number of terminals in graph is more than k, divide it for no more than k subgraphs with induced Steiner tree.

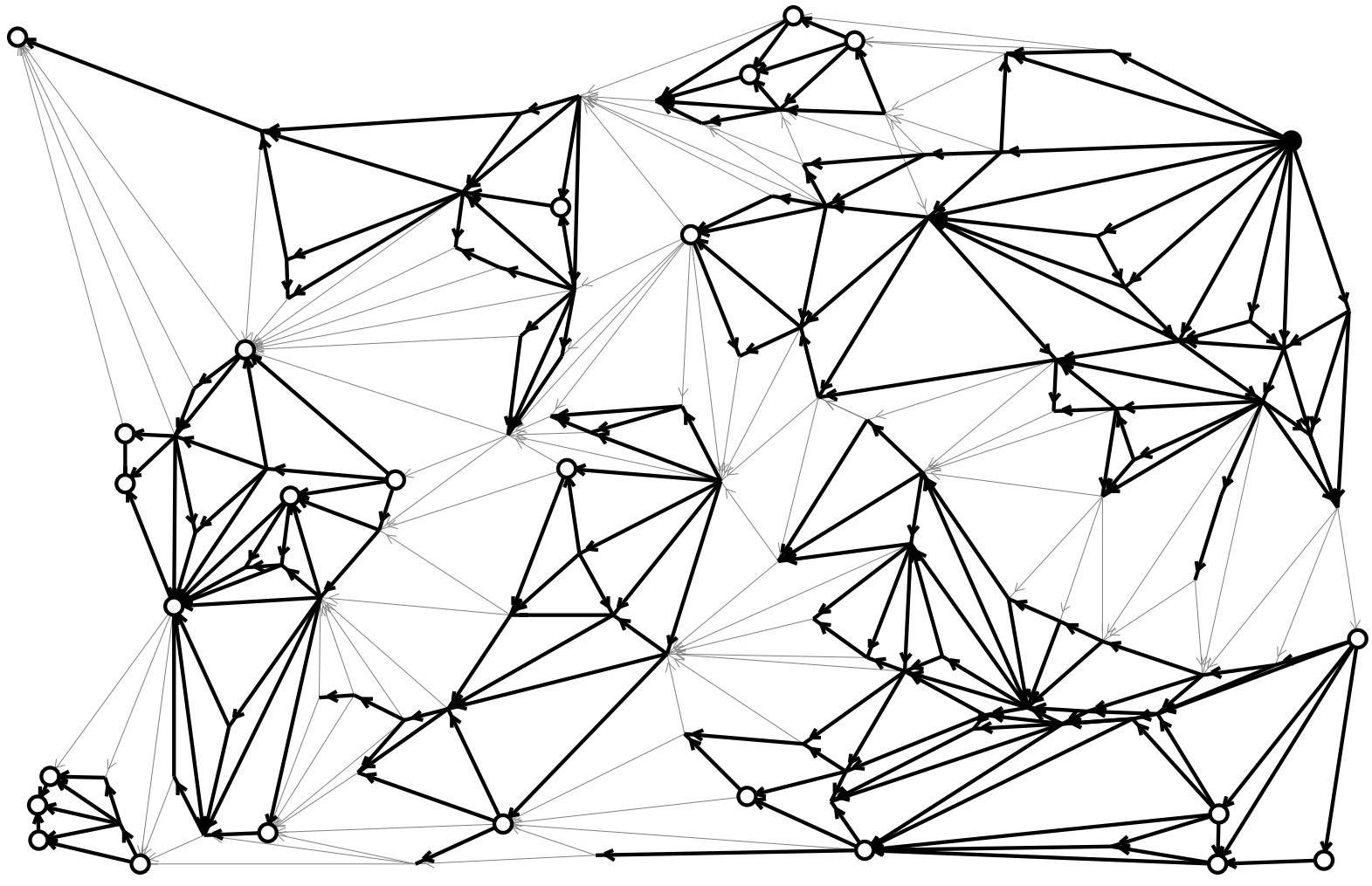
Dividing to clusters

Build auxiliary Steiner tree called *supporting*.

1. Find most remote from b branching node (call it b^*) and get subtree, that is rooted by it.
2. If terminals count in subtree is $> |E|/k$, pick subgraph as single cluster and cut corresponding subtree form supporting tree.
3. Otherwise, find on path $P[b, b^*]$ next most far from b branching node and examine it.

Add all nodes not included into any cluster to nearest clusters with pathes to them.

Dividing to clusters. Example



Constructing Steiner trees on clusters

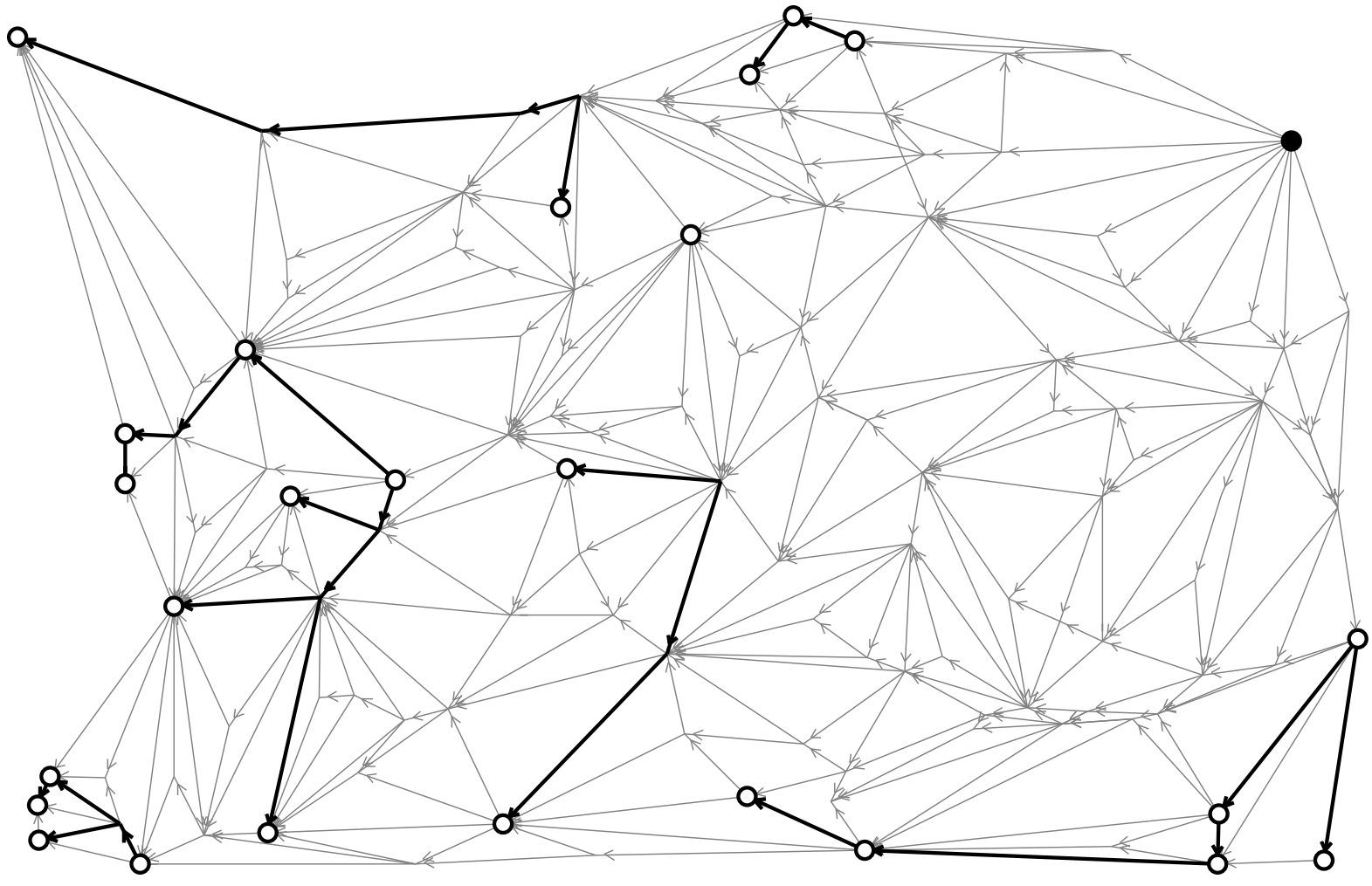
For clusters $C \in \mathcal{C}$ problem is posed:

- $E(C) = E \bigcap M(C)$,
- $b(C)$ — branching node, that is root of the cluster.

If $|E(C)| \leq k$ find exact solution using DP, else divide to clusters recursively.

Improvement: Drop off path from root of the tree to nearest branching node or terminal.

Constructing Steiner trees on clusters. Example



Finding Steiner tree on original graph

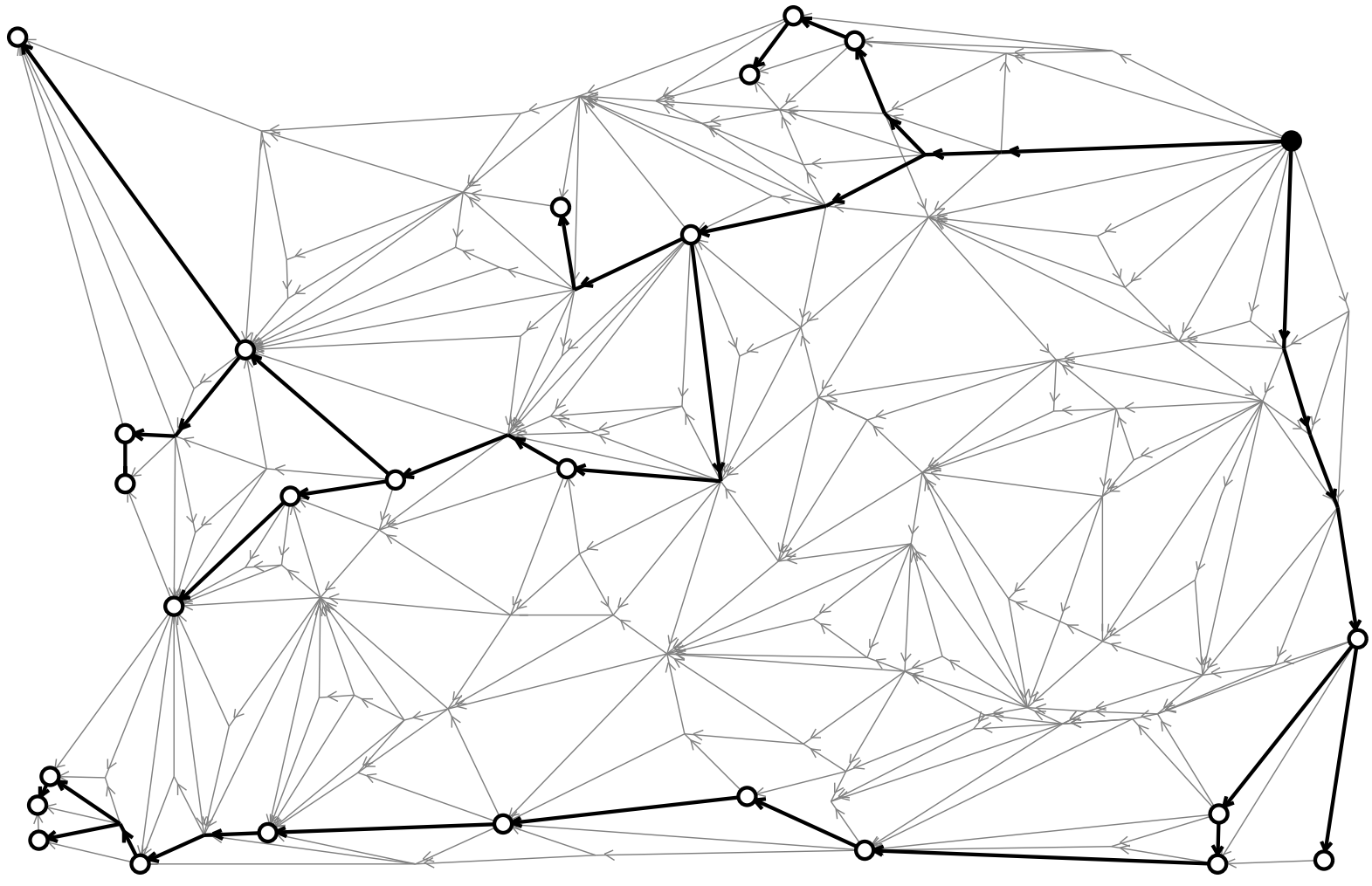
Modify original problem:

- Remove all arcs from $M \setminus M(C)$ to $M(C) \setminus \{b(C)\}$ for all $C \in \mathcal{C}$
- Assign zero length to all arcs, that are included in constructed partial Steiner trees.
- Terminal set consists of root nodes of partial Steiner trees(less than k nodes).

Local improvements:

- Consider set Y, consists of terminals and branching nodes of Steiner tree.
- For each node $y \in Y$ find nearest node $y^* \in Y$ at the path to b and remove from the tree $P[y^*, y]$
- Connect y and tree (connected component, that contains b) with shortest path.

Solution Steiner tree. Example



Algorithm complexity

Theorema 1

Computational complexity of k-cluster algorithm is equal $O(2^k tn \log m)$.

Lemma 1. *Computational complexity of constructing support tree is* $O(tn \log m)$.

Lemma 2. *Support tree* T_S *of graph* G*, constrained for each constructed with it cluster* G'*, equals to support tree* T'_S*, constrcuted on* G'.

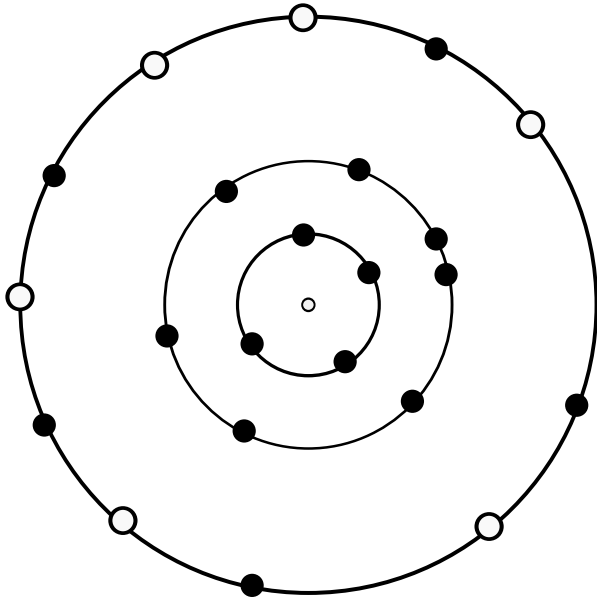
Lemma 3. *Let support tree* T_S *of graph* G *contains two branching nodes* b_1 *&* b_2*, besides* b_1 *&* b_2 *both lay on path from* b *to some leaf of* T_S*, and* b_1 *is closer to root than* b_2 *(there is path* $P(b_1, b_2)$*), then cluster induced by* b_2 *is subgraph of cluster induced by* b_1.

Lemma 4. *Let* $G' \in \mathfrak{C}$ *be a cluster on* G*. Then* $\mathfrak{C}(G') \subset \mathfrak{C}$.

Lemma 5. *Computational complexity of finding Steiner tree in k-cluster algorithm is* $O(2^k tn \log m)$.

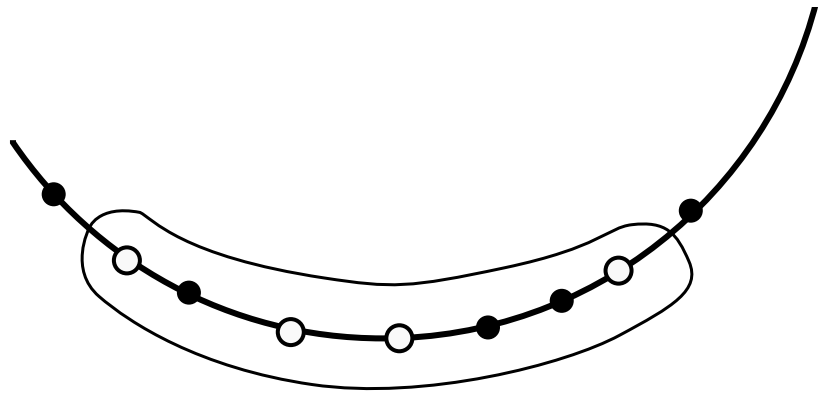
Lemma 6. *Computational complexity of local improvements is* $O(tn \log m)$.

Steiner tree problem in concentric circles



Modification: we can consider only subsets that consists of termnals sequentially placed on circle (not counting non-terminal nodes).

It is proven that the modified algorithm returns an exact solution of the problem.



Algorithm S^* for euclidean graphs

Essence of algorithm: constrain set of terminal subsets $\mathcal{R}$ in such way that $|\mathcal{R}| = P(|E|)$. Then solve it with DP.

$\mathcal{R} \neq 2^E$, than on the correctional step check $E_i \cup E_{i_0} \in \mathcal{R}$, and perform correcting only if true.

Graph is *euclidean*, if there are points on the plane correspond to nodes, and arc length equals to euclidean distance between corresponding points.

Naive method of constraint:

Let's call $J \subseteq E$ *b-acceptable*, if $\nexists m' \in E \setminus J : \exists m_1, m_2 \in J :$

$$\angle(m_1 b m_2) = \angle(m_1 b m') + \angle(m_2 b m'). \tag{7}$$

$\mathcal{R}$ — set of all b-acceptable sets.

Another methods of constructing $\mathcal{R}$

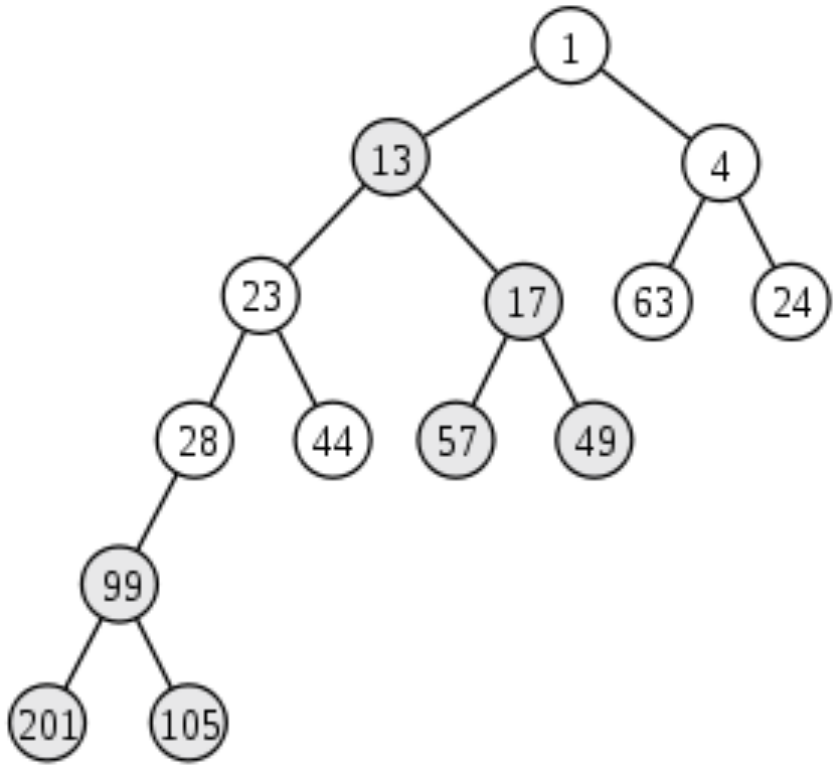
«Concentric circles» method: k iterations.
$T_i \subset E$ on i-th iteration ($T_0 = E$). All b-acceptable subset of T_i are added to $\mathcal{R}$. Average distance from b to terminals r^* is founded and T_i is splitted into two subsets — $T_{(i+1)_1}$ & $T_{(i+1)_2}$ ($\geq r^*$ и $< r^*$), and iteration repeats for both.

Generalized method: For each $m \in M$ subset fo terminals $T_m \subset E$ is founded that is reachable from m, and all m-acceptable subsets from T_m are added to $\mathcal{R}$.

Priority Queues

Has considerable impact on Dijkstra method and DP algorithm effectiveness.
Known implementations:

- Binary heap
- n-ary heap
- Binomial heap
- Fibonacci heap
- Buckets
- Leftist heap
- Skew heap
- Soft heap
- Brodal heap
- etc



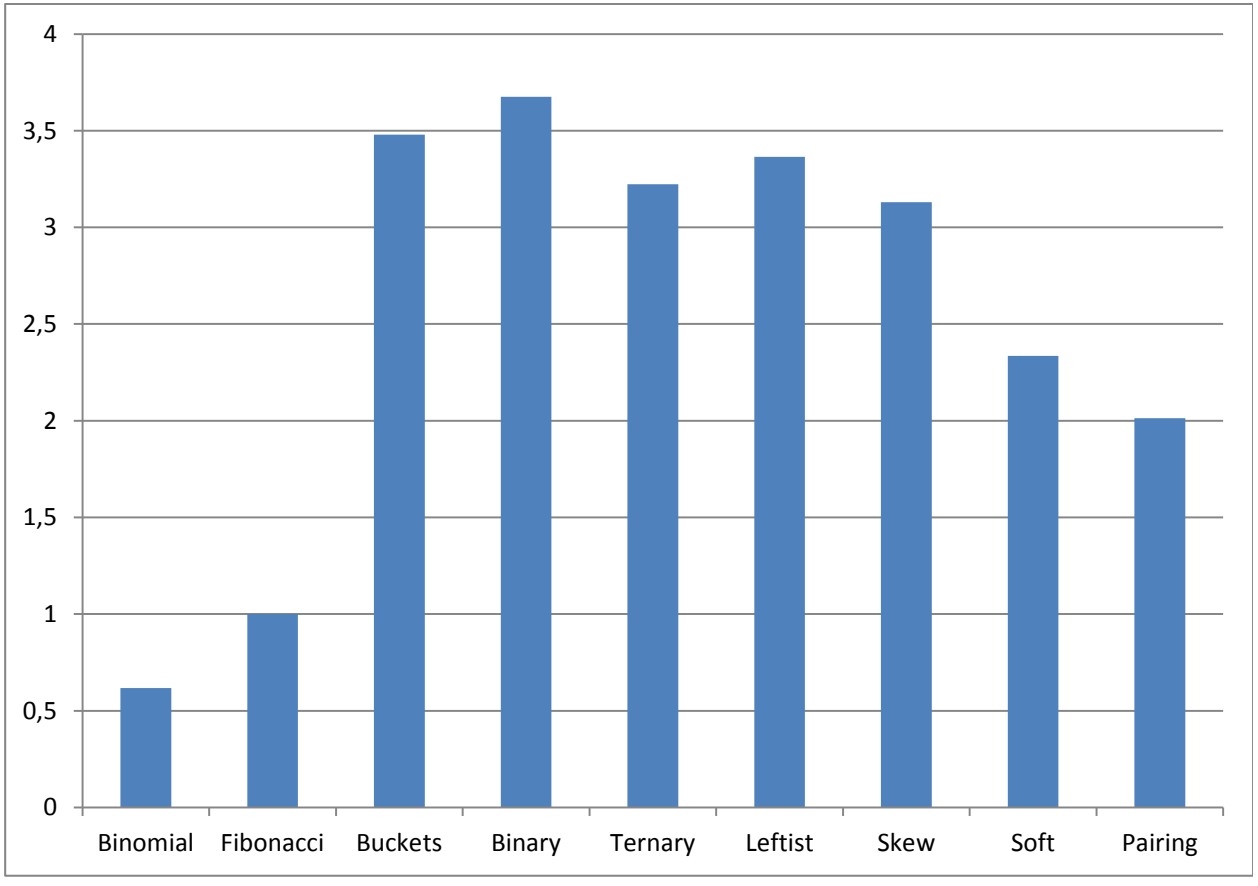
Priority Queues. Computational complexity

	Insert	**Extract**	**DecreaseKey**	**Dijkstra method**
Binary heap	$O(\log m)$	$O(\log m)$	$O(\log m)$	$O(n \log m)$
n-ary heap	$O(\log m)$	$O(\log m)$	$O(\log m)$	$O(n \log m)$
Binomial heap	$O(\log m)$	$O(\log m)$	$O(\log m)$	$O(n \log m)$
Fibonacci heap*	$O(1)$	$O(\log m)$	$O(1)$	$O(m \log m + n)$
Buckets	$O(1)$	$O(C)$	$O(1)$	$O(mC + n)$
Leftist heap	$O(\log m)$	$O(\log m)$	$O(\log m)$	$O(n \log m)$
Skew heap	$O(\log m)$	$O(\log m)$	$O(\log m)$	$O(n \log m)$
Soft heap*	$O(\log 1/\epsilon)$	$O(1)$	$O(\log m)$	$O(n \log m)$

In rows with * amortized run times are presented, worst-case in all other.

Comparison of priority queue implementations

Set of 400 problems c $|M| \in [80, 640]$, $|N| \in [120, 204480]$, $|T| \in [10, 160]$ from SteinLib.k-cluster algorithm.



Parallelization

Every step of iteration algorithm is separate problem.
Time of scheduling problem S_i: after correcting row for all disjunctive splittings of S_i.
Array of length $2^{|E|}$. i-th cell contains count of all possible disjunctive splittings of subset E_i. (For S is $2^{|S|-1}+1$).
At every row correction array cell is reduced by 1. If becomes 0, problem for corresponding subset scheduled for solving.

In k-cluster algorithm furthermore every problem is posed separately and can be solved simulataneously.

In algorithm S^* count of disjunctive splittings is founded by exhaustive search.

Other parallelization possibilities:

- Parallelization of Dijkstra algorithm
- Use of parallel priority queues

Experiments on parallel and sequential implementations

All tests performed on CPU with 4 physical cores and *hyperthreading*.

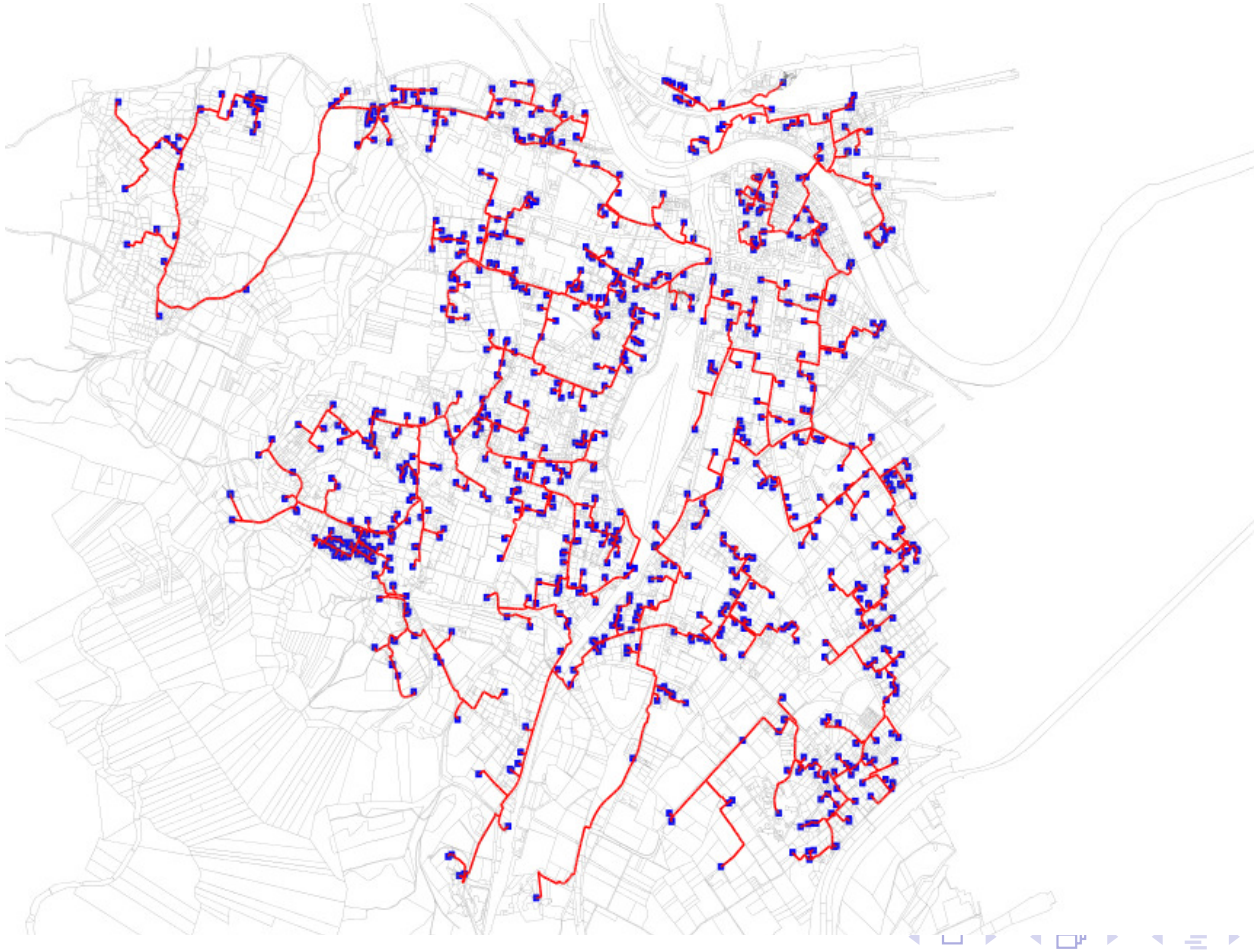
Exact: Set of 150 problems where $|M| \in [80, 640]$, $|N| \in [120, 204480]$, $|T| \in [9, 17]$. Parallel algorithm is up to 3.5 times faster than sequential.

k-cluster: Set of 400 problems where $|M| \in [80, 640]$, $|N| \in [120, 204480]$. Parallel algorithm is up to 2 times faster than sequential. On simple problems sequential mehtod is faster.

S^* algorithm Set of 150 problems where $|M| \in [160, 640]$, $|N| \in [2371, 204156]$, $|T| \in [20, 120]$. Parallel algorithm is up to 6 times faster than sequential.

Vienna Steiner Tree problem set

Issued in 2014. From real-world telecommunication networks. Between 42481 and 235686 nodes, 52552 and 366093 edges, and between 88 and 6313 terminals. Have subset of geo-problems, with node coordinates.



Some Vienna set problem solutions

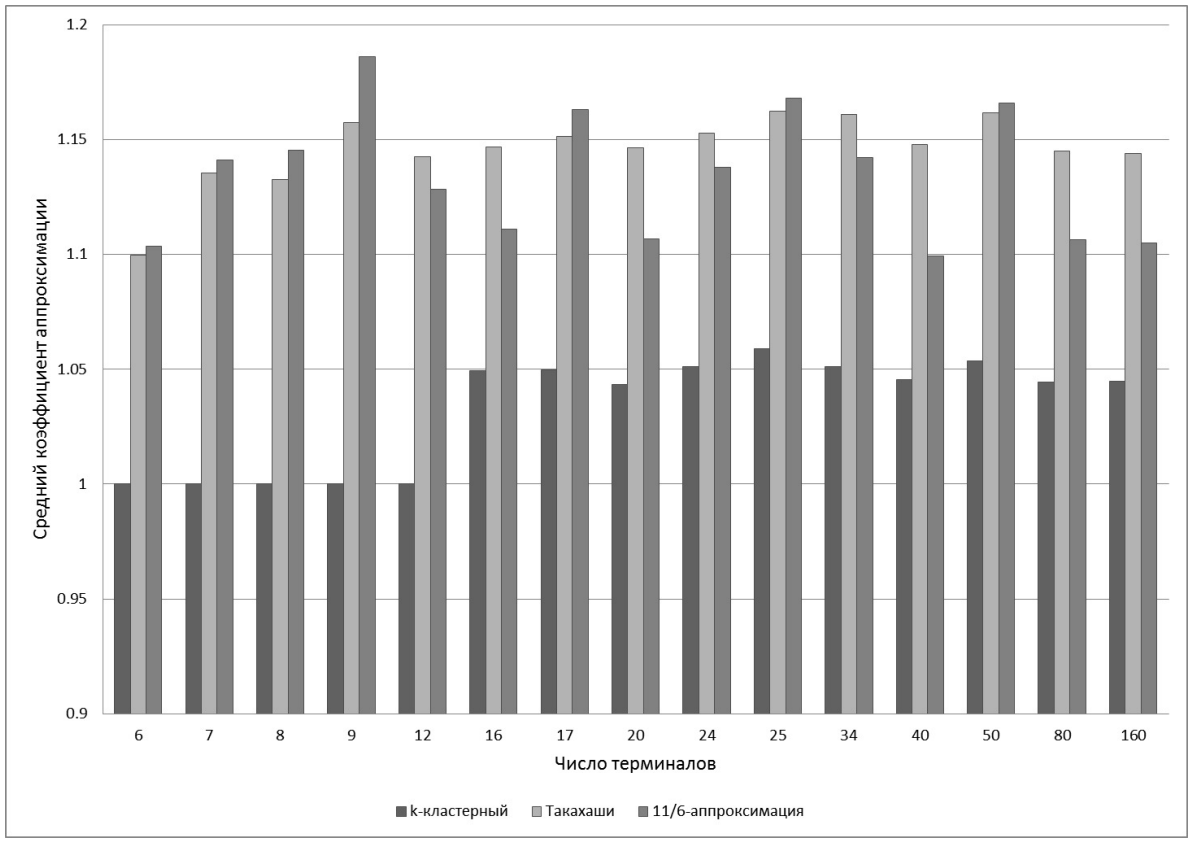
	Lower bound	Precise	Time (sec)	k-Cluster	Time (sec)	Precise gap	k-Cluster gap	DA gap
G101	3 439 226	3 604 599	86400	3 653 673	290	1,36%	6,24%	4,81%
G102	15 132 879	15 305 206	86400	15 770 592	7869	3,04%	4,21%	4,65%
G107	7 309 295	7 360 406	86400	7 793 398	1768	5,88%	6,62%	4,75%
G201	3 481 975	3 490 260	86400	3 656 831	265	4,77%	5,02%	3,86%
G202	6 849 281	6 852 245	86400	7 167 864	1509	4,61%	4,65%	4,25%
G204	5 313 548	5 313 548	34256	5 611 053	342	5,60%	5,60%	4,16%
G206	9 166 968	9 197 029	86400	9 478 401	1043	3,06%	3,40%	4,29%
G207	2 265 834	2 265 834	50754	2 307 000	225	1,82%	1,82%	3,09%
G304	6 629 770	6 896 969	86400	6 972 482	1278	1,09%	5,17%	4,03%

Thank you!

Comparison of k-cluster algorithm with other good-known

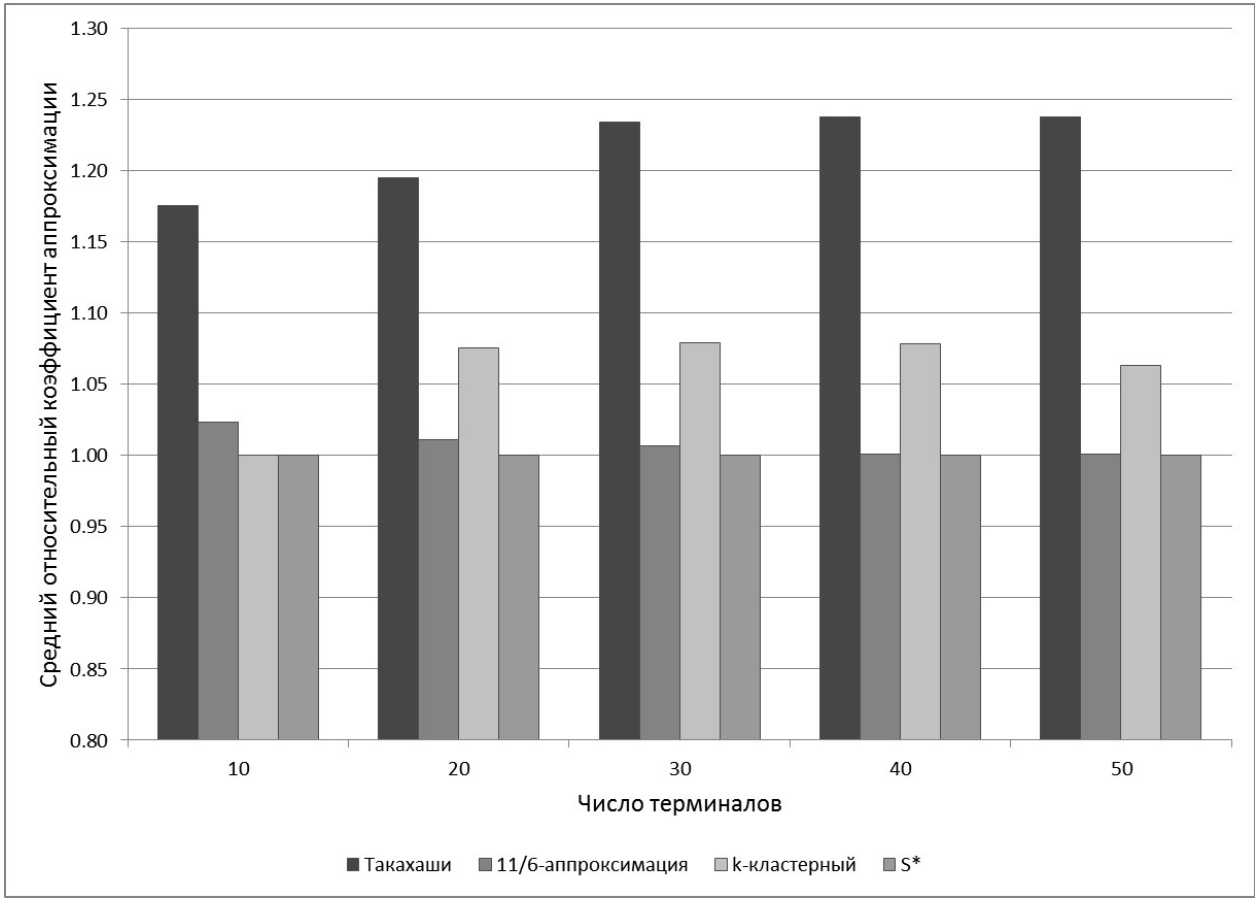
Set of 400 problems where $|M| \in [80, 640]$, $|N| \in [120, 204480]$, $|T| \in [6, 160]$ from **SteinLib** library. k_{approx} for k-cluster algorithm equals in average 1.05.

1. k-cluster
2. Takahashi-Matsuyama
3. Zelikovsky's greedy



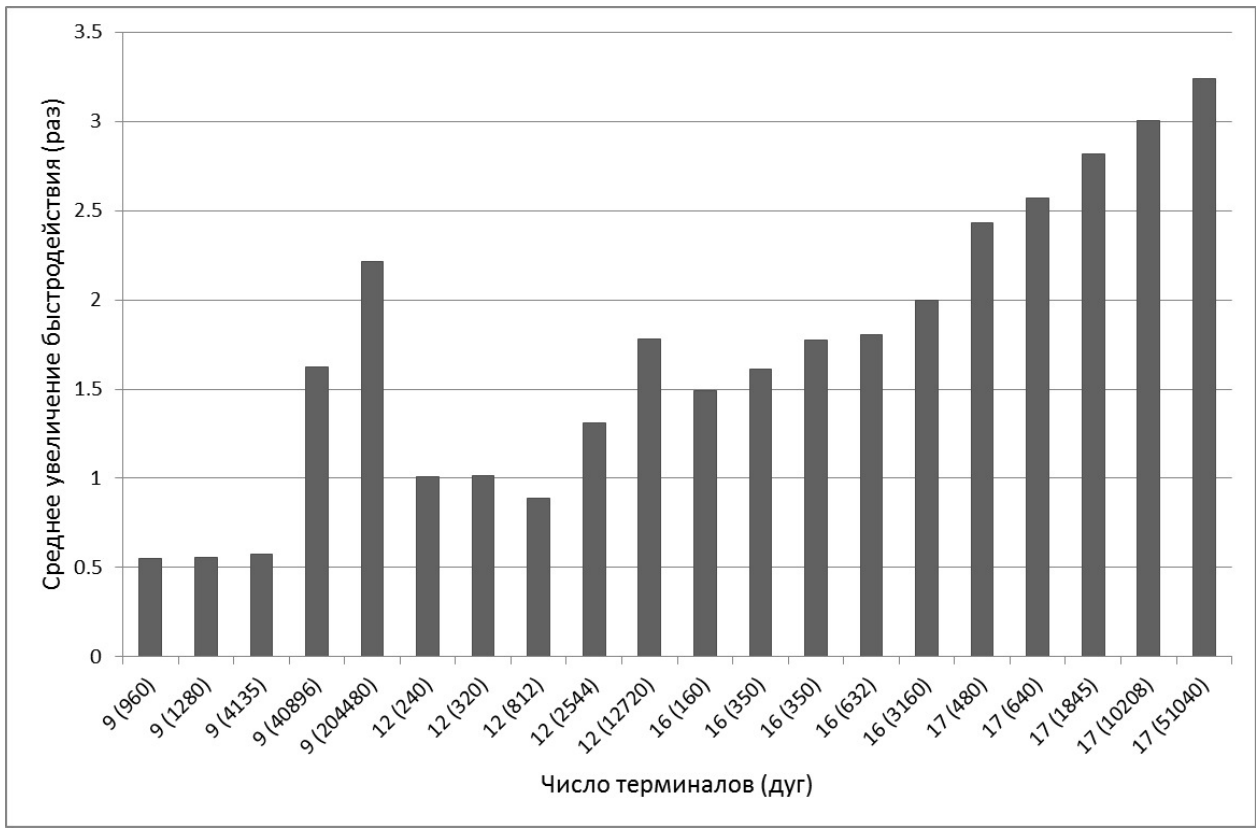
Comparison of S^* algorithm with some well-known ones

Set of 150 problems where $|M| \in [160, 640]$, $|N| \in [2371, 204156]$, $|T| \in [20, 120]$. Precision of solutions, getting with S^*, is the best.



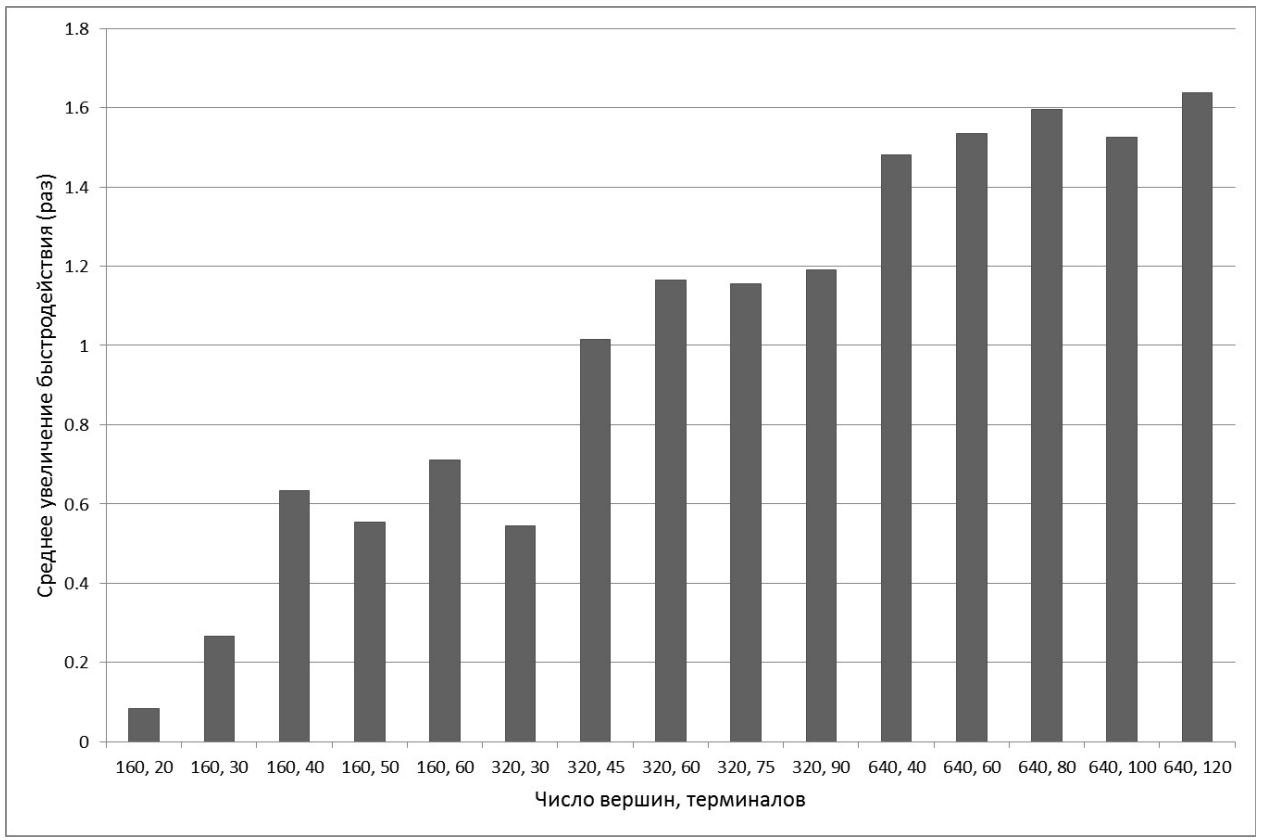
Comparison of sequential and parallel DP implementations

Set of 150 problems where $|M| \in [80, 640]$, $|N| \in [120, 204480]$, $|T| \in [9, 17]$.



Comparison of sequential and parallel implementations of k-cluster

Set of 400 problems where $|M| \in [80, 640]$, $|N| \in [120, 204480]$,



Comparison of sequential and parallel implementations of S^* algorithm

Set of 150 problems where $|M| \in [160, 640]$, $|N| \in [2371, 204156]$, $|T| \in [20, 120]$.