Building communication networks: on the application of the Kruskal’s algorithm in the problems of large dimensions

B F Melnikov¹ and Y Y Terentyeva²

¹ Shenzhen MSU – BIT University, 1 International University Park Road, Dayun New Town, Longgang District, Shenzhen, Guangdong Province, 518172, China
² Center of Information Technologies and Systems for Executive Power Authorities (Federal state Autonomous Research Institution), 19, Presnensky Val, Moscow, 123557, Russia

E-mail: terjul@mail.ru

Abstract. The paper deals with the development of the topology of ultra-large communication networks, i.e. networks containing several thousand vertices. In this case, the coordinates of the vertices of the undirected graph are somehow predetermined and a set of edges must be constructed. The main point of the options we are considering for developing the network topology is the minimum of the sum of weights of the edges; however, we note in advance that this criterion of minimality is often not the only objective function in the practical problems we are considering. In our previous papers, two realistically considered tasks were formulated. However, everything is not so simple, and we cannot use the direct version of Kruskal’s algorithm. The complexity of this algorithm depends on the representation of the data, i.e. the data structures used. In our situation (when the number of considered vertices is approximately 5000 to 10000), the operation of a simple version of the algorithm takes about a half an hour, which, of course, is acceptable for a one-time solution to the problem under consideration, but it is unacceptable in the case when such solutions are constructed repeatedly (in particular, iteratively). Some temporary improvements to the practical operation of the algorithm provide different options for using complex data structures. The subject of this article can be formulated as follows. We are moving from exact algorithms (in particular, Kruskal’s algorithm) to some heuristics. Moreover, for the starting problem that we are considering, we cannot work without heuristic algorithm at all. However, we describe two specific variants of a simple implementation of Kruskal’s algorithm for problems of large dimensions; in our titles, those are “the 1st and the 2nd algorithms of the usual implementation”. We have formulated two heuristics (“the 3rd and 4th algorithms”). In our opinion, one of these algorithms turned out to be quite acceptable; we present some practical results of computational experiments. And it is very important that these two heuristics will be useful not only for such a “0-th problem”, but also for much more complex problems.

1. Introduction
The paper deals with the development of the topology of ultra-large communication networks (i.e. networks containing several thousand vertices). In this case, as a rule, the coordinates of the vertices of all the network (undirected graph) are somehow predetermined (for example, as a point of a unit square determined by a pair of coordinates), and a set of edges must be constructed. The main point of the
options we are considering for developing the network topology is the minimum of the total length (the sum of weights) of the edges; however, we note in advance that this criterion of minimality is often not the only objective function in the practical problems we are considering.

In our previous papers [1], [2] etc., two realistically considered tasks were formulated. But, contrary to the seemingly natural sequence of publications, this paper does not consider in detail “one of the subsequent” problems (we have described at least 8 such new problems that we propose to formulate in the papers and implemented programs), but a “previous” one. This previous task is an implementation of Kruskal’s algorithm [3]–[5], and, of course, for labeled undirected graphs of very large dimensions.

However, everything is not so simple, and the “implementation complexity” (difficulties) is connected precisely with the estimation of the algorithm complexity. As with most graph theory algorithms, it is impossible to answer the question of the complexity of this algorithm, since the answer depends on the representation of the data (i.e. the data structures used). As is clear from what is stated in the preface, in our situation it is necessary to use all potential edges, the number of which (for N vertices) is equal to \( N^3 \). In this regard, enumeration (scanning) of all edges takes about \( N^2 \) steps, and such views are necessary \( N \). Thus, in our situation (when the number of considered vertices is approximately 5 000 to 10 000), the operation of a simple version of the algorithm takes about half an hour—which, of course, is acceptable for a one-time solution to the problem under consideration, but it is unacceptable in the case when such solutions are constructed repeatedly (in particular, iteratively, by a version of the brute force method).

Some temporary improvements to the practical operation of the algorithm provide different options for using complex data structures (and similar issues, among others, are also considered in this paper). For example, we can somehow store a certain number of unused edges of small length, and, if necessary, sort these edges, add new ones to them, etc. However, this approach is not a “panacea”, since in the worst case the complexity estimates (and the running time of the algorithms) are the same. All this formulates the need to consider and implement heuristic algorithms, instead of exact, exhaustive ones.

The whole subject of this paper can be formulated as follows. As we already said, in real situations, i.e. for our tasks, the simplest implementation of Kruskal’s algorithm is not very acceptable (in reality, the complexity of the algorithm turns out to be cubic, while we cannot store the weights of the edges of the graph in computer memory, since there will be more than 12 million of them for 5 000 vertices; and if and we can store it, it is hardly possible to constantly execute sorting algorithms for 12 million edge weights). In addition, we assume that the algorithm will work not once, but many times (because the user thought of everything to change the input data many times). All this is just another way of explaining that we are moving from exact algorithms (in particular, Kruskal’s algorithm) to some heuristics. Moreover, for the “zero” (starting) problem that we are considering (and as we have already noted, we have described at least 8 related problems) we cannot work without heuristic algorithm at all. However, we describe two specific variants of a simple implementation of Kruskal’s algorithm for problems of large dimensions (in our titles, those are “the 1st and the 2nd algorithms of the usual implementation”).

Therefore, for our problem, we have so far formulated two heuristics (in our names, “the 3rd and the 4th algorithms”). In our opinion, one of these algorithms turned out to be quite acceptable. And it is very important that these two heuristics will be useful not only for such a problem, which can be called “the 0-th problem”, but also for much more complex problems. About two of them, the so-called problems 1 and 2 (these numbers should not be confused with the numbers of algorithms), we have already written in [1].

The paper has the following structure. Section II contains not so much a description of Kruskal’s algorithm (we consider it to be widely known), but rather the authors’ thoughts on the possibility of its application in the problems we are considering. In Section III, we consider two variants of the usual implementation of Kruskal’s algorithm for high dimensional data. We also present the corresponding results of some computational experiments.
In Sections IV and V, we consider two heuristics which supplements the Kruskal’s algorithm. Contrary to our expectations, only one of them gives acceptable results. For all the Sections III, IV and V, we present the practical results of computational experiments.

In Conclusion (Section VI), we formulate some problems as a development of the topic we are considering. We propose to describe approaches to solving these problems in subsequent publications.

2. Preliminaries
This section contains not so much a description of Kruskal’s algorithm (we consider it to be widely known), but rather the authors’ thoughts on the possibility of its application in the problems we are considering.

It is well-known, that Kruskal’s algorithm is an efficient algorithm for constructing the minimum spanning tree of a weighted connected undirected graph. It is also used for some other related tasks. According to some sources, this algorithm is the same as Boruvka’s algorithm proposed in 1926.

In the usual version of Kruskal’s algorithm, it is necessary for a given weighted unoriented graph (its edges have weights) to construct a spanning tree of minimum cost (of minimum weight; both these formulations are used in the literature). The algorithm itself is extremely simple: we sequentially include in the constructed spanning tree that of the remaining edges that has the minimum cost; at the same time, we additionally make sure that a cycle does not form (as usual in algorithms for constructing a certain tree). It is easy to prove that as a result of the operation of the algorithm described in this way, the desired tree is obtained. Note also that we shall not distinguish between Kruskal’s algorithm proper and Prim – Jarník’s algorithm [6], since in our constructions the order of edge selection does not matter.

However, as we have already noted in Introduction, everything is not so simple, and the “implementation difficulties” are connected precisely with the estimation of the algorithm complexity; the answer to the complexity question depends on the presentation of the data. Since we do not distinguish between Kruskal’s algorithm proper and Prim – Jarník’s algorithm, any variant of working with an adjacency matrix leads to a formal estimate of the $N^3$ complexity. Therefore, as noted in Introduction, we use heuristic algorithms.

Among the heuristic algorithms, according to one of the possible classifications, we single out the so-called anytime algorithms. This is a special class of real-time algorithms, which at each certain moment of operation have the best (at the moment) solution, while the user can view these pseudo-optimal solutions in real time, and the sequence of such solutions in the limit usually gives the optimal solution. Note that such algorithms are needed in practice due to the fact that, as noted above, we run the program for our algorithm many times, and the final decision is made by a person, an expert. All this is in good agreement with one of the definitions of an expert system as a predictive system that includes knowledge about a certain weakly structured and difficult to formalize narrow subject area and is able to offer and explain reasonable decisions to the user (see [7] etc.); in our case, such a pseudo-optimal solution is some specific spanning tree of “small” weight, and the human expert decides whether such a tree is acceptable.

The construction of such algorithms is one of the tasks that can be united by a common topic with the approximate name “Training of fuzzy systems”. The authors, among other things, hope that the material of this paper formulates a method for constructing anytime algorithms for a whole class of problems, i.e. optimization problems of graph theory. It is also worth noting that the main goal of the paper is the practical issues of constructing algorithms, and not the creation of a corresponding theory.

In our tasks, the following two things are most important. First, such heuristic algorithms usually fit into the frame work of the general multi-heuristic approach to discrete optimization problems (see [8], [9] etc.) and therefore can be solved by approximately the same methods. Second, of course, the complexity of our heuristic algorithms is significantly less; in our case, it corresponds to $N^2$ (or slightly exceeds this estimate), which is quite acceptable; moreover, when a graph is represented as an adjacency matrix, it is unrealistic to expect better estimates. We note in advance that for the dimensions we are considering, we consider the solution time to be on the order of a few seconds, and not several tens of minutes.
3. Two variants of the usual implementation

In this section, we consider two variants of the usual implementation of the Kruskal’s algorithm for high dimensional data. We present the corresponding results of some computational experiments.

As we noted, we believe that Kruskal’s algorithm is widely known; therefore, we shall not describe the simplest version of its implementation at all. (With sufficient memory, the complexity of the implementation of this option is comparable to a student laboratory work.) The points are randomly thrown into the unit square (using a uniform distribution), and the choice of the next added edge occurs after iterating over all the edges not yet included in the graph. Of course, we make sure that a cycle does not form; for this, we stored the numbers of the current connectivity components of the created graph.

For the execution (when checking both this algorithm and all subsequent ones), a computer with a clock frequency of about 2.4 GHz was used. When checking, we tried to prevent other programs from running; it is clear that we cannot guarantee a very high accuracy of the obtained computation time values, but, apparently, very high precision is not needed. (Also apparently, there is no need to specifically launch the program without a working operating system.)

As the results of the work (table 1), we give only the time averaged over several (10–15) program launches; the precision is 3–4 significant decimal digits.

From the given temporal results of table 1, it is clear that it is impractical to reuse such an algorithm for the dimensions 5 000–10 000 of interest to us.

The modification of this algorithm is obvious (and, although the implementation is somewhat more complicated, it is also not much more complex in complexity than student lab work). In this modification, for each vertex we memorized the current version of the one closest to it (and not yet included in the generated graph), as well as the corresponding minimum distance. When a certain edge was included in the graph, we changed the indicated temporarily stored values, which in practice made it possible not to iterate over all possible edges every time. The calculation results are shown, and the meaning of the values is the same as the results of the simplest algorithm (table 1).

| Dimension | The simplest algorithm | Simplest algorithm with the temporary memorization | The first heuristic | The second heuristic | Badness (the first heuristics) | Badness (the second heuristics) |
|-----------|-----------------------|--------------------------------------------------|--------------------|---------------------|-------------------------------|-------------------------------|
| 1         | 512                   | 1.55                                             | 0.277              | 0.076               | 0.277                         | 1.16                          | >4                            |
| 2         | 1024                  | 12.4                                            | 1.30               | 0.269               | 0.312                         | 1.13                          | >4                            |
| 3         | 1536                  | 42.9                                            | 3.75               | 0.566               | 0.517                         | 1.11                          | >4                            |
| 4         | 2048                  | 101.3                                           | 8.23               | 0.979               | 0.925                         | 1.12                          | >4                            |
| 5         | 3072                  | 358.5                                           | 28.7               | 2.29                | 1.18                          | 1.09                          | >3                            |
| 6         | 4096                  | 1109                                            | 54.7               | 4.03                | 1.59                          | 1.11                          | >3                            |
| 7         | 6144                  | 140                                             | 6.17               | 2.02                | 1.10                          | 1.10                          | >3                            |

From the results it is clear that proportionality is not observed, but there is an expected improvement associated with the application of the memorized results; moreover, we can say that the running time of the algorithm does not much exceed the “quadratic” one. However firstly uncomplicated estimates at worst give the same thing as without applying the described improvement; secondly, the operating time is still long (recall that we need an algorithm and a corresponding program that will be launched many times – either by the user or by another program). Therefore, as we already noted in the introduction, our problem requires the use of heuristic algorithms.

4. The first heuristic (the third algorithm)

In this section, we consider the first heuristic which supplements the Kruskal’s algorithm. As in previous section, we present the corresponding results of some computational experiments. By our expectations, the application of this heuristic gives acceptable results.
The heuristic used in this algorithm is as follows. We form information in advance not about the only vertex closest to the one under consideration, but about several such vertices. In our program, for the total number of vertices from 512 to 8192 (6144 in the tables below), the numbers of the nearest vertices from 10 to 40 were used. Moreover, in order to reduce the total running time of the algorithm, we tried to connect some considered vertex with the nearest possible one (i.e. with one of those that do not form a cycle yet), choosing this new vertex from among those memorized in advance (their number, as we have already noted, is from 10 up to 40). In the event, that after the operation of the algorithm more than one connected component remained, we used the end of the algorithm, which actually coincides with the simplest version of Kruskal’s algorithm (i.e., with the first algorithm commented above).

As we already said, this heuristic did not show acceptable results. In table 1, in addition to the previous columns, we also indicated the average ratio (i.e. the average deterioration of the found answer); we call this average ratio by “badness”. A value not exceeding 1.5 would be quite acceptable here, and we have obtained better results. Besides, the execution time of the algorithm is very short. As before, the precision is 3–4 significant decimal digits.

Thus, on the dimensions we are interested in, we see an improvement in the execution time of the algorithm by about 20–25 times, while the deterioration in the quality of solution is acceptable values 1.10–1.12. Moreover, from our point of view, the stability of the badness values indirectly indicates that this algorithm is based on a fairly good idea.

5. The second heuristic (the fourth algorithm)
In this section, we consider the second heuristic which supplements the Kruskal’s algorithm. As in previous section, we present the corresponding results of some computational experiments. Unfortunately, contrary to our expectations, the application of this heuristic did not give acceptable results.

The heuristic used in this algorithm is as follows. We are doing an analogue of the algorithm for sequential exchanges of sorting an array. At the same time, for a certain step (which is usually equal to 1, but sometimes, if the selected random variable from a unit segment does not exceed 0.15, it can be increased to 8, see the program text below), we check whether the points (vertices) are located “closer” when exchanging in the array some two points lying at the distance of this step. Note that we are considering such “closurity” (“proximity”) for the entire set of points: the points lying nearby with adjacent indices in the array give the minimum “closurity”. After a certain number of executions of such exchanges (in the programs, we used from 5 000 to 30 000 for different variants of the dimension of the array of points), we get a “sorted” (by this feature) array of points.

After such a preliminary reshaping of the special array, we apply an algorithm that is somewhat similar to the previous one. However, we do not calculate (and, therefore, do not store) the nearest several points, but try to connect the connected components, including two points standing next to each other in the array.

Unfortunately, despite initial expectations, this heuristic did not show acceptable results. With a good running time (which, of course, was expected), the corresponding program gave poor results for comparing the objective function of the constructed graph with the optimal variant. Note also that the implementation of the “analogue of the sequential exchange algorithm” does not take much time, and, moreover, we make it independent of the dimension of the problem. Although, despite such badness values, the considered algorithm can be used in some problems (due to the “not very bad” badness values and very fast operating time). Besides, we may describe and implement some improvement of this algorithm.

6. Conclusion
So, we believe that as a result of the work done, we have obtained an algorithm that satisfies both the time and the goal criteria. Let us formulate some problems as a development of the topic we are considering. We propose to describe approaches to solving these problems in subsequent publications.
Here is a text from [8] about the generality of all discrete optimization problems we solve; including this text can be attributed to the problem considered in this paper. Each of these problems can be easily solved for small dimensions, but when passing to large dimensions, it is impossible to strictly solve them in real time even using the branch and bound method and other similar heuristics. For example, the traveling salesman problem (to which practically all the problems we consider can be reduced), even in a simplified setting, belongs to the class of NP-complete problems. The methods we describe for solving these problems are based on a combination of heuristics taken from several different areas of the theory of artificial intelligence. First, the unfinished branch and bound method is used. Secondly, to select the next step of this method in the presence of several heuristics, the dynamic risk functions are used. Thirdly, genetic algorithms are used simultaneously with the risk functions to select the averaging coefficients for various heuristics. Fourthly, simplified self-learning by the same genetic methods is used to start the unfinished branch and bound method. Then we can say that these combinations of heuristics represent a special approach to the construction of anytime algorithms for discrete optimization problems. Thus, one of the tasks for the nearest implementation is to apply the branch-and-bound method to the domain we have considered.

Separately arising tasks are associated with the special function of the second heuristic we have considered. For example, among them there is such a problem: is it always possible, in a certain number of steps, to obtain an array that has a minimum according to the criterion we described, or can we get either into a local maximum or into an infinite loop? And another related task: how to improve the last algorithm by improving the special function of the second heuristic itself?

The very first publication about Kruskal’s algorithm [3] used the title “traveling salesman problem”; at the same time, in those years, the corresponding terminology was not yet established. Indeed, according to the authors of this paper, many of the auxiliary algorithms used in the construction of a connected graph are similar to auxiliary algorithms used in the heuristic solution of the traveling salesman problem. At the same time, apparently, a much larger number of publications were devoted to the traveling salesman problem, and in this regard, the following question arises: which of the previously described methods for solving the traveling salesman problem can be applied in the subject area we are considering?

In some of our previous papers (see [10], [11] etc.), we solved so-called pseudo-geometrical traveling salesman problem. Accordingly, the question arises about the possible application of the algorithms from this paper to the data structures considered in those publications. We hope to return to some of these questions in our next publications. In addition, we are going to consider a significant increase in the dimension of the problems under consideration: from about 10 000 to about 1 000 000.

References

[1] Melnikov B, Meshchanin V, and Terentyeva V 2020 Implementation of optimality criteria in the design of communication networks Journal of Physics: Conference Series, International Scientific Conference ICMSIT-2020: Metrological Support of Innovative Technological 3095
[2] Bulynin A, Melnikov B, Meshchanin V, and Terentyeva V 2020 Optimization problems arising in the design of high-dimensional communication networks, and some heuristic methods for their solution Informatica and Communication I 34-40 (in Russian)
[3] Kruskal J 1956 On the shortest spanning subtree of a graph and the traveling salesman problem Proceedings of the American Mathematical Society 7(1) 48-50
[4] Cormen T, Leiserson C, Rivest R and Stein C 2001 Introduction to Algorithms (Second Edition. MIT Press and McGraw-Hill) ISBN-262-03293-7
[5] Goodrich M and Tamassia R 2006 Data Structures and Algorithms in Java (Fourth Edition. John Wiley & Sons, Inc.) ISBN 0-471-73884-0
[6] Prim R 1957 Shortest connection networks and some generalizations Bell System Technical Journal 36(6) 1389-401
[7] Russell S and Norvig P 1995 Artificial Intelligence: A Modern Approach (Simon & Schuster)
ISB 978-0-13-103805-9

[8] Melnikov B 2005 Discrete optimization problems – some new heuristic approaches Proceedings – Eighth International Conference on High-Performance Computing in Asia-Pacific Region HPC Asia Beijing 73-80

[9] Melnikov B, Radionov A, Moseev A and Melnikova E 2006 Some specific heuristics for situation clustering problems Proceedings – 1st International Conference on Software and Data Technologies ICSOFT Setubal 272–9

[10] Melnikov B and Romanov N 2001 Once again on the heuristics for the traveling salesman problem Theoretical problems of computer science and its applications 4 81-8 (in Russian)

[11] Makarkin S and Melnikov B 2013 Geometric methods for solving the pseudogeometric Version of the Traveling Salesman Problem Stochastic Optimization in Computer Science 9(2) 54-72 (in Russian)