Software for solving of TSP

ROMAN LUKATSKY, VLADIMIR RYBINKIN
Bureau of Internet Technologies BIT Ltd
Novoposelkovaja str., 6/7, Moscow, Russia, 123459
RUSSIA

Abstract: - This paper is devoted to investigation of problems of discrete or combinatorial optimization on the example of symmetric travelling salesman problem. A parametric algorithm for solving of TSP from very fast to exact, including the version with mobile nodes is offered. The developed software was tested by solving of TSPs that best-known solutions are already known.

Key-Words: - Travelling Salesman Problem, TSP, DBMS, Graph

1 Introduction

The travelling salesman problem (TSP) [6] is a problem in discrete or combinatorial optimization. It is a prominent illustration of a class of problems in computational complexity theory, which are classified as NP-hard. At two hundred years history, numbers of algorithms offered for its solution are estimated in thousands. There are various kinds of heuristics, genetic algorithms, simulated annealing, Tabu search, ant system, etc [2].

The elementary algorithm guaranteeing a finding of the exact solution is a full enumeration of all variants. For travelling salesman problem the number of possible tours is so great, that by this way the solution cannot be found at reasonable restrictions in time already at several tens of nodes. The branch-and-bound algorithm and its numerous versions [8] a little improve situation, as in this case many variants can be rejected without search if it is possible to prove, that in a rejected array of variants the solution cannot be present. The existing algorithms allows raising dimension of a problem approximately up to 100 nodes, and in some cases up to much lot [1] [3]. Nevertheless, the labour input of calculation of the solution by this method, as well as by brute force algorithm, has exponential complexity.

Most of methods, developed up to present tense, are suitable for very limited number of nodes [4], and exact solutions are found for graphs only up to several tens thousand of nodes (13509, 15112, 24978, 33810), at that the processor time is estimated as tens of years. Moreover, how it is possible to tell about **exact** solutions, if distances round off to integer, i.e. are calculated **approximately**.

The main criteria of quality of algorithms are accuracy of the received solution and the time for its revelation. As additional criteria one can be considered the expense of hardware resources, influence of initial data on accuracy of solution, simplicity of software realization, etc.

We offer our own algorithm of search of TSP solution and based on graph DBMS [9] the adaptive software for its realization. The developed software should be able to find solutions for TSPs, stored in graph database within a reasonable time with possibly low error. Parametrical adjustments should allow focusing the software on achievement of the different purposes: search of exact or approximate solution, quick search of fairly good solution for very larger graphs, etc. This exploration was carried out within the limits of research of opportunities of graph DBMS.

The purposes of this paper is an examination of created software on well-known TSPs to test its applicability for solving of different kinds of travelling salesman problem and an attempt to use it for solving TSP on large and huge (millions of nodes) of graphs.

2 Course of exploration

It is evidently that for construction of the shortest tour it is not obligatory to repeat a way of salesman: it is possible to consider nodes and edges of the graph in arbitrary order, say, all simultaneously. The approach of iterative change of current tour seems to us successful. One of the popular methods, using this approach, is stated in [7]. This method in some development provides very good results [5]. We also suppose that at any moment there exist some solution (current tour). At start we simply connect the first node with the second, the second - with the third, the last - with the first. We'll name the shortest tour as "best" and any shorter than current tour as "desired".

Let's suppose that we already know the best tour. It is obvious that any not optimal current tour can be reformed to the best tour by replacing no more than in extreme case all tour edges: for example, on fig.1 are presented the best tour ABCDEFGHA and the current tour ABHCDFEGA. We'll spend the replacement consistently, since some node (node A) that has "old" (dotted line), not included into best tour, an edge AG.

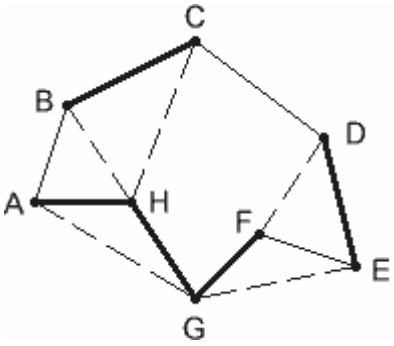


Fig.1. Detour of the contour.

We delete the edge AG from the tour and insert the "new" (thick line) edge GF that is entering into the best tour from the second node G of old edge AG. The second end of new edge necessarily point to node that has one or two old edges. As this node F becomes "overloaded" by edges, one of its edges should be removed (edge FD). As in each node from which old edges leave, the same quantity of new edges comes, so process of consecutive replacement of old edges on new ones come to the end with locking of a contour. Certainly, having begun with any of nodes having an old edge, we shall bypass all contour. Generally such contours can be more than one and detour of some of them can lead to temporary crippling of the tour.

It is obvious that the sum of lengths of new edges for all contours is less than the similar sum of deleted edges, i.e.

$$1+2+3+\ldots +k > 1'+2'+3'+\ldots +k' \quad (1)$$

Thus, there always exists at least one contour for which edges rightly similar inequality and at least one order of its detour when this inequality is correct on each step of detour. Differently, from any current tour it is always possible to create the best tour at keeping of benefit on each step of replacement of old edges on new ones. Therefore, the search of the best tour consists in revealing of contours and of continuously profitable order of detour of contours and also of nodes in each of them. Unlike [5] our algorithm do not stop until exact solution.

2.1 Algorithm for TSP

Let's consider on a concrete example step-by-step action of algorithm based on above principles. Each edge of current tour ABHCDFEGA (fig. 2) is consistently checked at each of its ends on existence of shorter edges that are not including into the tour yet. For example, an edge GE (fig. 2a) has three shorter edges: GF, GH and DE. Each of them is consistently checked on an opportunity of inclusion in tour instead of deleted edge GE. If we lead the detour from node G towards to node E, owing to selected direction of detour, we must insert the edge DE (fig. 2b). Then one of edges of overloaded node D is deleted from the tour (edge DF) and next new edge FG is inserted. We shall note that after this operation we have already received shorter tour ABHCDEFGA. Of course, the sum length of new and old edges must be profitable. Recursively continuing the search of the best tour by replacement already not two but three edges (fig. 2c), we again shall receive shorter tour. All next attempts of replacement of edges are unsuccessful.

On prima facie, at worst labour input of search of the solution by the offered method is even worse than that of brute force. However this laboriousness strongly depends on configuration of current tour. In particular, if current tour pass on sides of regular polygon, there is no shorter edge in the graph than an edges of the tour (at observance of triangle inequality), and the same algorithm will come to end without any recursive call, i.e. will have the polynomial (quadratic) complexity. An efficiency of an inequality (1) as criterion of termination of depth first search directly depends on current tour length (beta-boundary). The reduction of beta-boundary sharply abates amount of computations and accordingly raises the efficiency of the offered approach. As each new tour is always shorter than current tour, we'll replace the current tour on any new immediately to decrease the labour input of following computations.

Search of way from current tour to desired it is advisable to carry out with minimal calculations.

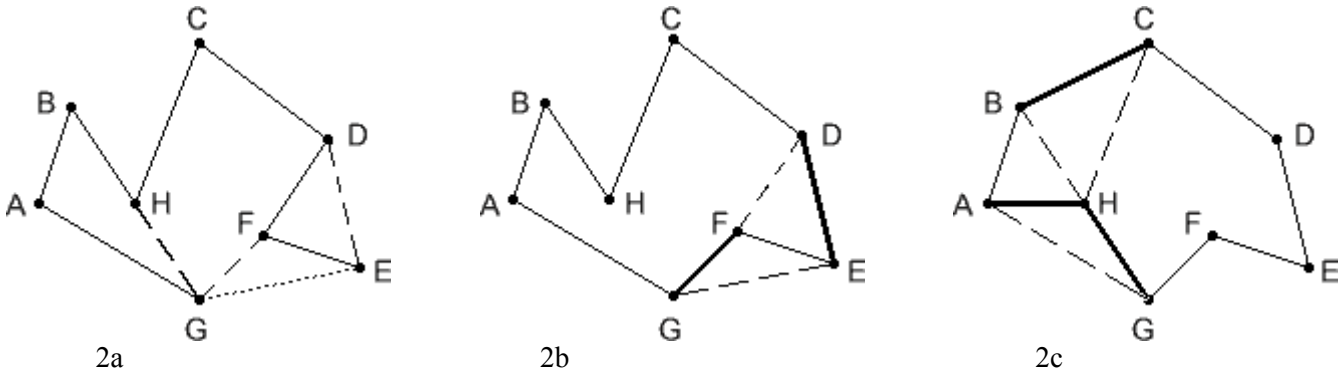


Fig.2. Step-by step algorithm of detour.

Table 1. Experimental results.

Range (nodes)	Number of graphs	Start		Fast		Depth 2		Stop	
		Mean (%)	Max (%)	Mean (%)	Max (%)	Mean (%)	Max (%)	Mean (%)	Max (%)
1-100	16	398.8	799.3	92.5	265.4	7.6	20.5	0	0
101-1000	43	727.3	3289.2	38.7	119.3	11.5	23.5	0.7	3.4
1001-10000	25	1559.2	4225.3	22.5	58.5	16.1	28.8	3.9	4.9
>10000	6	5888.7	9281.8	38.2	82.6	15.8	18.8	4.2	4.9
Total	90	1244.1	9281.8	43.7	119.3	12.4	28.8	1.4	4.9

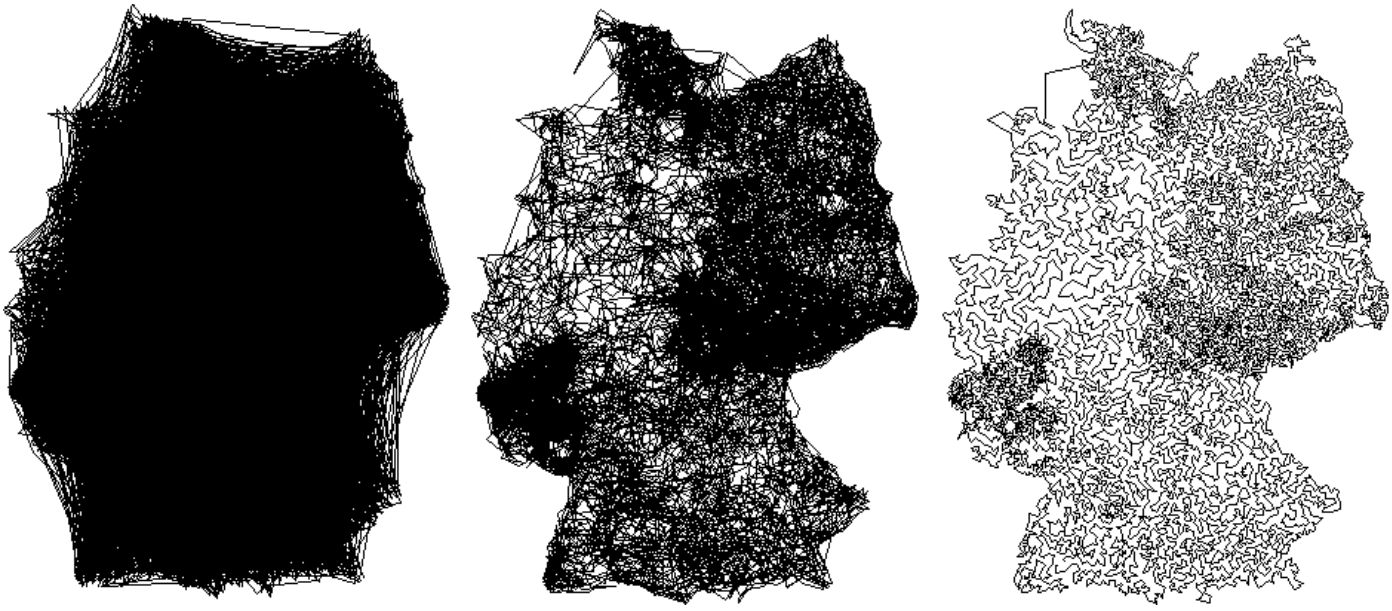
Obvious way of decrease of processing time for recursive algorithms is the cascade increasing of recursion depth. Therefore we shall carry out the search iteratively, by replacing on each iteration minimally possible quantity of edges. Differently, simultaneous replacement of k edges must be made only in case of impossibility of tour improvement at previous iterations with smaller quantity of edges (1, 2, 3, … k-1). Inasmuch as replacement of one edge is allowable only in case of presence of multiple edges that is impossible in classical statement of TSP, we shall raise the depth of recursion consistently, starting with 2. The above algorithm have been programmed and checked on real graphs for TSP.

2.2 Benchmark data and tools

Original data and exact solutions were received from site of symmetric TSP [10]. Computer Pentium-4, 2.4 GHz (512 Mb RAM), Windows-2000, the programming language C were used. Compilation was carried out by means of compiler BC ++ v3.1.

2.3 Experiments

To check the quality of created algorithm we solved some symmetric TSPs with already known exact solutions [10]. Initial tours have been created by consecutive connection of nodes according to increasing of their indexes. At that such initial tours for some graphs already have an error in a few percents or less (for instance, for the graph pr2392 the initial tour already represents the exact solution). Hence for detailed checking of algorithm at various states of current tour we created an additional initial tour by connection of nodes in correspondence with ascending of their coordinates, and took for processing worst of two ones. As calculation by declared algorithm is executed right up to revealing of exact solution that can demand heavy expenses of processor time, computations were artificially interrupted after 4 hours if the algorithm did not finish earlier of this time by acknowledged receipt of the best tour (burma14, ulysses16, eil51, st70, eil76, rat99 - 6 problems altogether). Results are shown in table 1.



Start: 7039.5% After 1 minute: 171.3% After 4 hours: 4.9%

Fig.3. Solving of TSP for d15112

Table 2. Experimental results for top 50 graphs.

Graph	Start, %	Fast, %	Depth 2, %	Stop, %	Length	Graph	Start, %	Fast, %	Depth 2, %	Stop, %	Length
pr264	298,2	30,6	23,5	0,5	49378	rl1323	1256,4	27,3	20	2,3	276476
a280	239,3	15,5	10,7	1,4	2614	nrw1379	1157,7	18,1	14,4	2,4	57990
pr299	404,9	26,1	11,8	1,1	48715	fl1400	3607,1	9,9	9,9	2,3	20583
lin318	1020,7	22,4	15,3	0,9	42428	u1432	164,4	14	11,3	2,8	157301
rd400	1310,6	107,4	13,4	1,4	15498	fl1577	3489,6	16,7	15,8	3	22926
fl417	3289,2	12,7	11,3	0,1	11878	d1655	459,9	18,3	16,2	2,2	63507
gr431	614,6	19,9	8,5	1,5	173902	vm1748	2872,9	45,9	12,8	3,7	349062
pr439	411,7	26,3	15,4	0	107255	u1817	408,7	26,5	21,3	3,5	59180
pcb442	470,2	24,2	13,1	1,6	51599	rl1889	1985,5	58,5	15,4	3,8	328527
d493	509,9	22,1	14,5	1,6	35545	d2103	680,4	31	28,8	2,2	82240
ali535	1565,6	42,9	14,4	0,4	203225	u2152	373,1	22	20,6	3,5	66487
u574	629,9	17,6	13,3	2,4	37797	u2319	107	7,2	4,5	0,8	236136
rat575	1292,2	21,6	16,9	1,7	6890	pr2392	1451,3	18,2	15,7	3.7	392157
p654	251,4	13,6	9,6	0,1	34669	pcb3038	1356,6	16,5	13,6	2,9	141693
d657	738,8	18,8	14,4	1,7	49751	fl3795	4225,3	19,7	19,1	3,9	29898
gr666	1139,5	15,3	10	3,4	304491	fnl4461	3116,5	17	14,9	2,5	187203
u724	554,4	22	16,4	1,7	42639	rl5915	2226,3	25,7	21,6	4,9	593039
rat783	1504	22,5	17,8	1,7	8960	rl5934	1977,7	27,9	22,2	4,8	582941
dsj1000	2888,4	94,9	14,3	1,3	18894359	pla7397	1910,6	14,8	12,7	3,6	24104074
pr1002	671,5	21,9	12,2	1,6	263073	rl11849	9281,8	82,6	18,8	3,6	956265
u1060	703,7	16,4	13,2	2,5	229664	usa13509	7861	18,1	15,4	4,6	20895464
vm1084	2136	33,6	12,5	2,5	245314	brd14051	4925,3	16,5	15,1	4,1	488619
pcb1173	940,8	12	11,1	3,8	59040	d15112	7039,5	72,4	14,4	4,9	1650770
d1291	420,2	17,5	17,4	3,8	52711	d18512	5248,3	15,4	14,6	3,8	669600
rl1304	1279,9	26,1	24,9	3,8	262641	sw24978	976,3	24	16,5	4,5	894130

Apparently from the table that algorithm finds quite acceptable solutions for reasonable time. For 32 graphs from 90 investigated (and for all graphs with quantity of nodes up to 150 inclusively) the length of final tour is equal that of best-known tour. In other cases an error is always less than 5%. Any dependence of quality of the solution on initial data is not revealed. Detailed results for 50 graphs with the greatest quantity of nodes are shown in table 2.

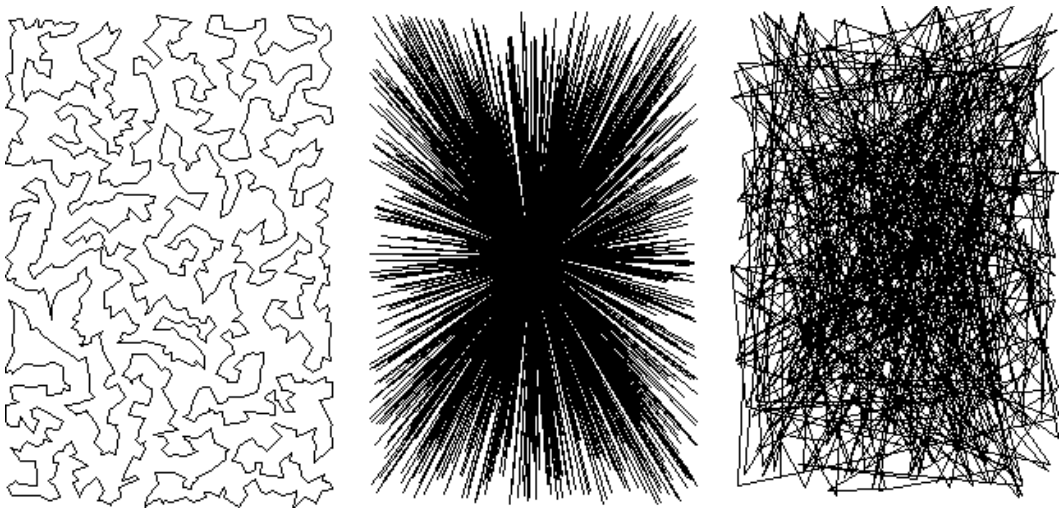
2.4 Processing of very large graphs

Usually it is considered, that the opportunity of using of polynomial-time algorithms allows solving TSP. However for larger graphs, with many millions of nodes, any algorithms except linear-time, as a matter of fact, appear useless. We shall note that observance of triangle inequality is not obligatory. Whether the search of good enough tours at this case is possible?

We modified our algorithm (fig. 2) a little as follows. Search of edges for replacement we shall make not among all edges of tour, but only among k first of them - of nearest neighbors of a current edge in tour. On the first iteration we accepted k = 1, but only for those edges which length exceeds average length of an edge of current tour. For long edges (twice exceeding average length) the value k is doubled, and for superlong (four times and more) is doubled once again. On the contrary, for short edges the value k accordingly decreases. Thus, longer edges are examined more closely, and short, less than average length of current tour, are not considered at all. It is obvious, that such edges in any tour there is more than half, therefore labour input of work of algorithm on the first iteration appears even below linear on number of nodes of the graph.

At next iteration the value k is doubled, i.e. the algorithm smoothly "flows" from linear to quadratic complexity. Transition to new iteration is carried out, however, not always but only in case the previous iteration cannot essentially improve current tour. Otherwise, if on current iteration it is revealed a significant amount of new tours (we have set their amount as 1/32 or more from number of nodes), it repeats at the same value of k. Work of high-speed algorithm comes to end, when k will exceed a quarter from number of nodes of the graph.

Experiments have shown, that application of the declared high-speed algorithm allows to lower a total operating time of first iteration in tens times. The beta-boundary even for very far from optimal tours was



1. Length 128795.59 2. Length 4716223.02 3. Length 330898

Fig.4. Solving of TSP in special cases

sharply decreased (fig. 3). The length of tour after the termination of work of high-speed algorithm in most cases makes up 20-30% from optimum, so it is already possible to recommend it for practical utilization. On usual PC software allows to find a satisfactory solutions at reasonable time for graphs with number of nodes up to some millions (maximal size of the really tested graph was 123456789 of nodes).

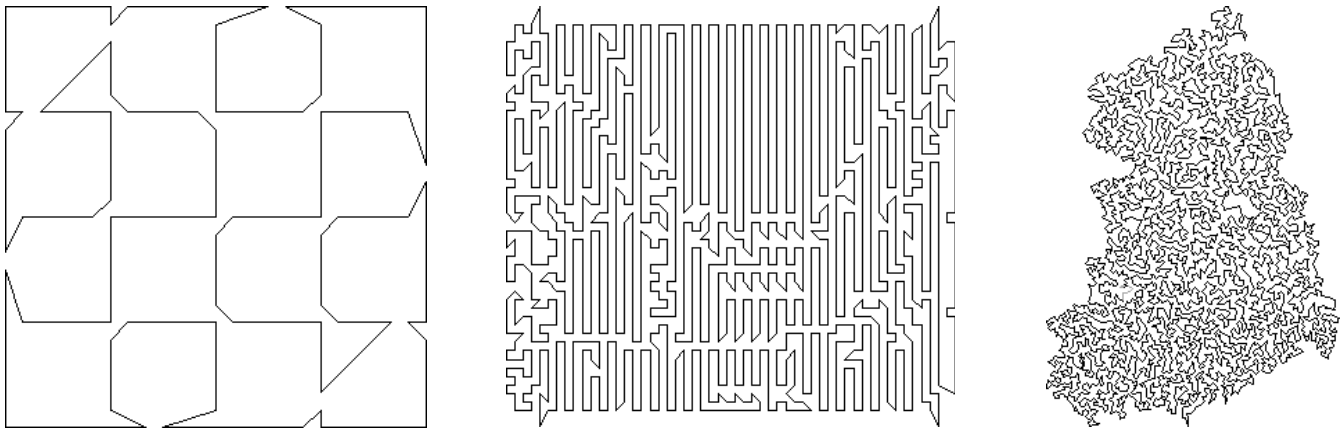
Various variants of solutions for the graph of 1234 nodes are shown on fig. 4: the shortest tour (1), the longest tour (2) and the shortest tour with wrong triangle inequality (3). In last variant the length of edges between two nodes was calculated as product of their indexes. Despite of "awful" view, the initial length of tour was decreased more than on two orders.

2.5 Specificity of search of exact solution

The idea of increasing the speed of search of exact solution is simple enough: at "heavy" iterations the only task is to find though any solution, to "tousle" the current tour, not doing any attempt to improve the found solution. However after each "heavy" iteration an attempt of optimization of current tour on depth 2 is done. Besides already after the termination of the first iteration there remain in tour, as a rule, only 30-40 % of "interesting" edges (that demand a recursive call). After each next iteration the appreciable part of "interesting" edges becomes "uninteresting" as all deeper iterations terminate not on recursion depth, but "naturally" (if losses as a result of distortion of current tour will exceed the maximal expected benefit). Therefore on each new iteration there remains less and less edges for consideration and, despite of increase of recursion depth, the time spent on new iteration sometimes even decreases a little.

As it has been told above, exact solutions, as a matter of fact, are not exact in reality, as they minimize the length of tour together with errors of rounding off. This "petty fraud" is not harmless at all, as rounding off of distances sharply (it is better to say "radically") decreases the labour input of calculations, say, by branch-and-bound method. Many routes which after rounding off of distances up to integers become equal, at really exact calculations will have different length, and cannot be excluded from consideration any more. Differently, the exact solutions for graphs with tens thousand of nodes will demand not decades but millennia of processor time. One of advantages of our approach is that accuracy of calculations practically does not influence on effectiveness of algorithm.

As the created software is suitable for checking of existing solutions (for this purpose it is enough to set such solution as initial tour), it has been revealed, that many exact solutions can be "improved", if length of edges do not round off to integers. So, it has been shown that for graphs eil51, st70, eil101, ch130, ch150 and others the found tours, strictly speaking, are not optimal. In particular, 9 solutions for the graph d15112 (profit 0,00027%) have been found. It is easy to be convinced of correctness of the solution: it is enough to replace a final chain of the author's solution 5755-12438-4501-11824-13059 on 5755-11824-12438-4501-13059 or to replace 4757-10600-8259-7265 on 4757-8259-10600-7265, etc. The author's solution of the graph sw24978 nodes also it has appeared possible "to improve": 147 tours, which length less resulted as the best (maximal profit 0,00159%) have been found. Correctness of the solution also is easily to check by replacement of a chain, for example, 483-501-530-546-502-484 on 483-530-501-502-546-484.



ts225 u2319 fnl4461

Fig. 5. Some examples of TSP solutions

3 Discussion of results

The offered algorithms show quite satisfactory quality of the solution and high enough efficiency for graphs of large dimension and arbitrary configuration. They are suitable for revelation of both approximate and exact solution, at that the productivity of receiving of approximate solution is commensurable with that of heuristic algorithms [5]. The software can be easily modified for graphs with incomplete or superfluous number of edges, for asymmetric TSP. Speed of search of the exact solution can be considerably increased by using of multiprocessor hardware as the algorithm can be modified for parallel architecture on arbitrary number of computers and practically without loss of efficiency. Some examples of TSPs solutions are shown at fig. 5.

Our software is rather unpretentious to available computing resources. It can work even at the processor 8086 and 640K RAM. Nevertheless, in such conditions a number of nodes can reach 1000 and more. Fast speed of algorithm allows using it for solution of TSP even if all nodes of the graph move. The version of software for dynamic TSP is presented in [11].

As the graph and current state of calculation is kept in a database, calculations can be interrupted at any moment to continue it later, at that the solution is the best tour found to the moment of a stop. This circumstance allows using our software for improvement of the tours received by other (heuristic) algorithms or for the proof of their optimality. Even more interesting use of a DB consists in an opportunity of storage of a tree of variants of current calculation and its reduction to cyclic graph. Thus all permutations of profitable pairs of edges "new-old" are reduced to their combinations that, certainly, should reduce extremely the search time. Unfortunately, this problem demands deeper algorithmic study, and is not realized within the limits of this paper.

4 Conclusion

The experimental results show that the offered algorithms have competitive potential for solving discrete optimization problems.

References:

[1] D.Applegate, R.Bixby, V.Chvatal and W.Cook. Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problems, *Math. Program, no. 1-2, Ser. B, 91-153, ISMP*, 2003

[2] N.Biggs, E.LLoyd and R.Wilson. Graph Theory 1736-19366, Clarendon Press, Oxford, 1976.

[3] Concorde TSP solver for Windows *http://www.tsp.gatech.edu/concorde.html*

[4] L.Fang, P.Chen, S.Liu. Particle Swarm Optimization with Simulated Annealing for TSP. 6th WSEAS International Conference AIKED'07, 2007

[5] K.Helsgaun. An Effective Implementation of K-opt Moves for the Lin-Kernighan TSP Heuristic, *Roskilde University*, 2006

[6] E.Lawler, J.Lenstra, A.Khan and D.Shmoys. The travelling Salesman Problem: A Guided Tour of Combinatorial Optimization *John Wiley & Sons*, 1985.

[7] S.Lin, B.Kernighan, An effective heuristic algorithm for the traveling-salesman problem. *Operations Res. 21, 498-516*, 1973

[8] H.Perez, J. Salazar-Gonzalez. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics, vol. 145*, 2004.

[9] V.Rybinkin, R.Lukatsky. Elastic model for processing of heterogeneous data, *WSEAS transactions on systems and control, Vol.2*, 2007

[10] Symmetric travelling salesman problem, *http://www.iwr.uni-heidelberg.de*

[11] Symmetric travelling salesman problem, demo version of software for graphs with mobile nodes, *http://www2bit.ru/d_tsp.zip*