RESOURCE CHARACTERISTICS OF WAYS TO ORGANIZE A DECISION TREE IN THE BRANCH-AND-BOUND METHOD FOR THE TRAVELING SALESMEN PROBLEM

Mikhail V. ULYANOV
Leading Researcher, V.A. Trapeznikov Institute of Control Sciences,
Russian Academy of Sciences;
Professor, Software Management Department,
National Research University Higher School of Economics
Address: 65, Profsouznaya Street, Moscow, 117997, Russian Federation
E-mail: muljanov@mail.ru

Mikhail I. FOMICHEV
Student, Software Engineering Program,
National Research University Higher School of Economics
Address: 20, Myasnitskaya Street, Moscow, 101000, Russian Federation
E-mail: mikhail.fomichev94@gmail.com

The resource efficiency of different implementations of the branch-and-bound method for the classical traveling salesman problem depends, inter alia, on ways to organize a search decision tree generated by this method. The classic «time-memory» dilemma is realized herein either by an option of storing reduced matrices at the points of the decision tree, which leads to reduction in the complexity with additional capacity cost, or matrix recalculation for the current node, which leads to an increase in complexity while saving memory. The subject of this paper is an experimental study of temporal characteristics of solving the traveling salesman problem by the branch-and-bound method to identify a real reduction of span time using additional memory in a selected structure of a decision tree. The ultimate objective of the research is to formulate recommendations for implementing the method in practical problems encountered in logistics and business informatics.

On the basis of experimental data, this paper shows that both considered options of the classic algorithm for the traveling salesman problem by the branch-and-bound method generate software implementations with an exponential dependence on the execution time of the input length. The experimental results permit us to suggest that the applicability of an additional memory capacity of no more than 1 GB results in a significant (up to five times) reduction of the time span. The estimate of the resulting trend makes it possible to recommend practical application of the software implementation of the branch-and-bound algorithm with storage of matrices - with a really available 16 GB random-access memory and with limitation of the expected average computation time of about one minute on modern personal computers whereby problems having a dimension no more than 70 can be solved exactly.

Key words: traveling salesman problem, branch-and-bound method, time and memory efficiency, data structure, experimental research.

Citation: Ulyanov M.V., Fomichev M.I. (2015) Resource characteristics of ways to organize a decision tree in the branch-and-bound method for the traveling salesmen problem. *Business Informatics*, no. 4 (34), pp. 38–46.
DOI: 10.17323/1998-0663.2015.4.38.46.

Introduction

A large variety of practical settings in the area of business informatics and logistics comes down to the classical traveling salesman problem. The abundance of heuristic methods for its solution does not mean rejection of obtaining exact solutions for this problem. Obviously, for precise methods having an exponential complexity the researchers would like to have estimates of the dimension of problems which can be solved within a reasonable time, as well as modified exact algorithms with a better time efficiency.

For certain algorithms, an increase of time efficiency can be achieved by using an additional amount of available memory. This situation arises in algorithms allowing us to replace re-computations for storing previously obtained intermediate results. This approach can also be used for algorithms implementing the branch-and-bound method for the traveling salesman problem. By the example of modifications of the classical algorithm for the exact solution of the traveling salesman problem using the branch-and-bound method proposed by J.D.C.Little, K.G.Murty, D.W.Sweeney and C.Karel [1], this paper illustrates a possible increase of the time efficiency using an additional memory to store matrices in the data structure of the search decision tree.

1. Setting up the traveling salesman problem

The problem description lies in the fact that the salesman has to visit a number of cities for product advertising and selling purposes. It is assumed that between each pair of cities there is an internal transportation. Let us call these visiting rules a tour in which each city is visited only once. The fare between the cities is known, and, in general, the return fare is different. The challenge is to find the traveling salesman tour with minimal cost. It is obvious that the number of tours is finite, and we can solve the problem by straightforward enumeration. But, unfortunately, a complete set will contain (n — 1)! tours, for this reason an exhaustive solution algorithm for solving the real dimensions of the problem becomes totally unacceptable.

Setting up the problem in terms of the graph theory is built by association of cities with the graph nodes, and the transport routes and fare with loaded arcs. We get a complete directed asymmetric graph at n nodes without proper loops prescribed by the cost matrix $C=(c_{ij})$. The problem is called symmetric if the fare between any two cities does not depend on the direction, and asymmetrical, if that is not the case. The absence of proper loops can be denoted as $c_{ii}=\infty\ \forall i=\overline{1,n}$. Then, the problem is formulated as a problem of finding a covering (complete) cycle at the lowest cost, which is called a tour, on a complete directed graph specified by asymmetric (in the general case) cost matrix C [2].

Let us also enter the problem definition in Euclidean integral-valued space E_z^{n-1} [3]. It is based on the concept of permutation on a set of integers. Let us further represent through $\pi(k,l),\ l\geq k$ a set of all permutations of integers (it is obvious that a number of such permutations is $(l-k+1)!$) and consider a set of permutations $\pi(2,n)$, which contains $|\pi(2,n)|=(n-2+1)!=(n-1)!$ of various permutations (exactly the number of various tours is available in the traveling salesman problem with n cities). Since a tour is a complete cycle across the nodes, the initial node of the tour can be chosen arbitrarily. Let us fix a node number 1, and assume that some permutation $\mathbf{x}$ from a set $\pi(2,n)$ orders the tree traversal starting from the first, and after the last point given by this permutation, we return to the beginning of the tour. Therefore, the components of vector $\mathbf{x}$ in space E_z^{n-1} are simply ordered numbers from two to n, and the vector is associated with some permutation from $\pi(2,n)$, and it can be written:

$$\mathbf{x}\in\pi(2,n), x_i\in\{2,...,n\},\ i=\overline{1,n-1},\ x_{ij}\neq x,\ with\ i\neq j \quad (1)$$

Let us note that the sum of squares of the components of different vectors $\mathbf{x}$ is the same: these are different permutations of numbers from two to n. Thus, the end points of various vectors $\mathbf{x}$ are points on the positive hemisphere in E_z^{n-1} centered at the origin and with a radius of

$$r=\sqrt{\sum_{i=2}^{n}i^2}=\sqrt{\frac{n\cdot(n+1)\cdot(2n+1)}{6}-1},\ \mathbf{x}\in S_z^{n-1}(0,r).$$

Let us further determine the function

$$N\times N\underset{c}{\to}R^+,(i,j)\to c(i,j),(i,i)\to\infty$$

which assigns the value of the edge to each ordered pair of graph nodes, incident to this pair. In the constructed formalism, the traveling salesman problem in space E_z^{n-1} has the following statement:

$$\begin{cases} f(\mathbf{x})=c(1,x_1)+\sum_{i=1}^{n-2}c(x_i,x_{i+1})+c(x_{n-1},1), f(\mathbf{x})\to min \\ \mathbf{x}=(x_1,\cdots,x_{n-1})\in\pi(2,n)\end{cases} \quad (2)$$

The branch-and-bound method under further consideration works, if not explicitly, with this statement of the traveling salesman problem.

2. Description of branch-and-bound method for the traveling salesman problem

The general idea of the branch-and-bound method is a separation of the entire set of feasible solutions into subsets to further reduce the enumeration - a branching procedure. Every such subset shall be linked to an estimate (lower bound in the minimum search), providing a truncation of those subsets which intentionally do not contain an optimal solution – this is a procedure for constructing bounds. Therefore, the method results in investigation of a tree solution space model. A set of all salesman tours in which the objective functional (2) is minimized, specifying the tour cost, is such an initial subset in the problem under investigation. The ideas presented below from the algorithm authors [1] are a kind of classic of the branch-and-bound method. To construct the algorithm, two basic procedures, i.e. branching and bounding, are proposed. Let us consider the branching process [1]. The construction of a search decision tree starts with the root, which will correspond to a set of all tours, i.e., a root of the tree is set R of all $(n — 1)!$ tours possible tours in the problem with n towns. The branches going out from the root are determined by selecting one edge, for example, arc (k, l). The idea of the algorithm authors [1] is to divide the current set of tours into two subsets: one which most likely contains the optimal tour, and the other one which most likely does not contain this tour. To do this, a special algorithm for selecting arc (k, l) is proposed, which likely is a part of the optimal tour. The set R is divided into two subsets $\{k, l\}$ and $\left\{\overline{k,l}\right\}$. The subset $\{k, l\}$ includes all tours from R containing arc (k, l), i.e. passing through it, and the subset $\left\{\overline{k,l}\right\}$ includes tours that do not contain this arc. Let us note that the idea of the algorithm authors is very promising if the branching process is organized so that at each step the «right» edge is selected, the entire process will be completed after n steps. A fragment of such tree is shown in *Fig. 1* (the figure is taken from paper [2]).

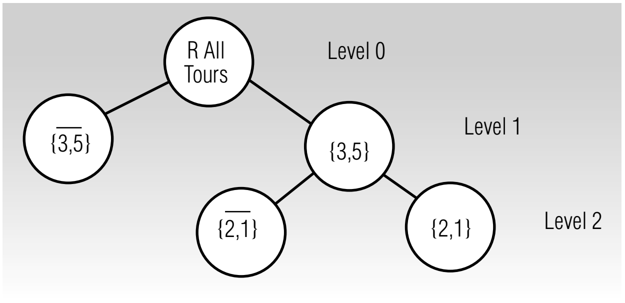

Fig. 1. Fragment of a search decision tree

The lowest cost limit of any tour of this set is associated with each tour of this set. It is obvious that the problem is to obtain the most accurate lower bounds. The reason for that is as follows. Let us assume that a specific complete tour T with cost $s(T)$ has already been obtained. If the lower bound associated with a set of tours represented by a search tree node is higher than $s(T)$, then until the end of the search process this and all subsequent points are not to be considered. In implementation this leads to truncation of the search decision tree by shedding all leaves of the search tree having a value higher than $s(T)$. A detailed presentation of other steps of the method can be found, for example, in papers [2, 3].

3. Diagram of the branch and bound method for the traveling salesman problem

Let us present the following diagram of the branch-and-bound method (B&B) for the traveling salesman problem, in which the following notations are introduced [2]. Let X be the current top of the search tree, and (k, l) be an edge across which the branching occurs. Let us denote the tops immediately following X through Y and $\overline{Y}$. The set Y is a subset of tours from X passing via arc (k, l), and set $\overline{Y}$ is a subset of tours from X not passing via arc(k, l). Let us denote the calculated the lowest bounds for sets Y and $\overline{Y}$ by $w(Y)$ and $w(\overline{Y})$ respectively. Let us denote the cheapest tour known to the algorithm at the given moment by z_o, provided that at the time of initialization is $z_o = \infty$ [1].

A(C, n) Diagram of the branch and bound method for the traveling salesman problem

```
(n - dimensions, C – cost matrix.)
1. Initializaiton.
2. Reduction of cost matrix C.
3. Setting a root of the search decision tree X=R
Reduction of the initial matrix – calculation w(X).
While (w(X) < z0)
    begin
        4. Selection of branching node (k, l).
        5. Branching process. Creating top Y¯
        and calculation w(Y¯).
        6. Branching process. Creating top Y
        and calculation w(Y).
        If (the order of cost matrix in top Y = 2)
            then
                begin
                    7. Performance of exhaustive estimation for top Y
                    If (w(Y) < z0)
                        then
                            begin
                                z0 = w(Y)(store the tour)
                            end
                end
        8. Selection of the following point of the search decision tree
        and setting X
        9a. Calculation of matrix fragment C,
        corresponding to selected top X,
        based on a route from the root to the current point.
(or)
        9b. Reading of matrix fragment C,
        corresponding to selected top X,
        from the structure of storage of the search solution tree.
    end (while w(X) < z0)
Optimal solution with cost z0 is not found.
End.
```

For the purposes of this article, let us draw our attention to some stages in this enlarged scheme.

Step 1. Selection of a data structure for storing the search decision tree is of interest here. At the same time, it is necessary to bear in mind that a number of the search tree points can be significant.

Step 3: Initialization of the search tree root. The main question is whether the cost matrix will be stored together with the top of the tree. An alternative option is to recalculate the cost matrix for the current top based on the original one. This is a classical choice between performance and memory requirements. The classical algorithm requires recalculation of the matrix costs for the newly determined current point of the search decision tree.

Step 9. Since the node of the search tree *X* in step 8 has already been selected as a current one, the task of this step is to obtain a cost matrix corresponding to top *X*. If we keep the truncated cost matrices together with the nodes of the search tree, the matrix already exists (9b). Otherwise, we need to find a path from the root to this node and consistently adjust the original cost matrix (9a), i.e. re-compute it for this point.

Discussion of algorithm specifics. What results in terms of complexity can be expected for different cost fixed-dimension matrices? It is known and suggested that the complexity of all more or less efficient algorithms implementing the branch-and-bound method for the traveling salesman problem in the worst case is exponential. This assumption is based on the fact that the traveling salesman problem is *NP*-hard and, therefore, any exact algorithm of its solution has an above-polynomial complexity. In the best case, if the dimension of the cost matrix keeps declining, the estimate of complexity is polynomial. This is obvious, because the assessment of complexity of each internal step (4-9) of the polynomial algorithm by a linear dimension *n* of the cost matrix, and, in the best case, the main loop is executed no more than *n* times, because the tour consists of *n* arcs. Therefore, the spread of expected execution time with a fixed dimension of the cost matrix *n* is very high and depends on the numerical values of its elements. This is an example of quantitative parametric algorithm, with a strong parametric dependence; the algorithm belongs to the class *NPRH* [3]. Theoretical analysis of the expected complexity for a particular entry based on a preliminary study of the cost matrix is very complicated and often goes beyond the scope of our analytical capabilities.

However, the time efficiency will be determined specifically as a selected structure to store the search decision tree, and a decision in step 9 with storage or recomputation of the current cost matrices. The matter is that an average number of active nodes grows exponentially. The experimental data from [4] gives an experimentally obtained approximation $R(n) = 3,6932 \cdot e^{0,1819n}$. In this regard, the effectiveness of elementary operations with the decision tree (add, delete, select) determines the time effectiveness of the software implementation of the algorithm as a whole [5, p. 130-131].

4. Data structures for the search decision tree

In order to select the data structures for the search decision tree, let us look at the three best-known options of the tree organization [6, 7], i.e. a binary heap, red-black tree and AVL tree

Table 1 shows the complexity of operations (information is taken from [6, 7]), which are performed above the search decision tree in the branch-and-bound method for the traveling salesman problem. It is evident from the table that the operating time (asymptotically) of the insert and element removal operations for all three data structures are equal. However, the minimal element search operation for the binary heap is performed over a constant time, but for the AVL and red-black trees over a logarithmic time. Moreover, in the functions of complexity of these operations coefficients with the term with higher exponent of the polynomial are much higher for red-black and, especially, AVL trees than for the binary heap, because additional operations are spent on the tree balance. It is also worth noting that three data structures have the same requirements for storage *O*(*n*). Therefore, it may be assumed that to store decision tree leaves a binary bunch will be suitable structure as compared to others, although this issue will be studied additionally and is beyond the scope of this study.

Table 1.

Complexity of operations for various ways of tree organization

Reference line	Binary heap	Red-black tree	AVL-tree
Complexity of insert	$O(log(n))$	$O(log(n))$	$O(log(n))$
Complexity of remove	$O(log(n))$	$O(log(n))$	$O(log(n))$
Complexity of search minimal element	$O(1)$	$O(log(n))$	$O(log(n))$

On the basis of the data presented by the authors, a decision was taken in this study to use a binary heap as the structure for storing the search decision tree.

5. Objectives of the experimental research

In this experimental research, the authors set the following objectives. The main objective is to illustrate a possible increase of the time efficiency through the use of additional memory for storage matrices in the search decision tree structure. The issue concerning the effect of matrix storage in the nodes of the search tree on the exponent in a trend of the average calculation time or only on a multiplicative constant in the trend function is of additional interest. Secondary objectives were to study a dependence of the memory capacity on the entry length in the comparative analysis of experimental results and the data obtained in paper [4], the distribution of the relative frequencies of the observed times, and to obtain data on the scope of variation and sample standard deviations for characteristics under study and design average time prediction for longer entry lengths.

6. Description of the experiment plan and hardware

Consideration was given to an asymmetric traveling salesman problem. The cost matrix was generated by a pseudorandom uniform generator standard for C++ pseudo-uniform generator. In order to reduce the total time of the experimental research, the cost matrix elements had an integer type, and a range of generation of oriented edge weights was chosen from 1 to 2^{15}. For the same purposes, the range of values of input lengths of the traveling salesman problem (linear dimension of the matrix values) from 25 to 45 with an increment of 1 was selected.

Algorithm [1] was implemented (with no prediction of the final tour of the greedy algorithm) with the structure of storing the search decision tree in the form of a binary heap in the classical implementation with recalculation of the cost matrix (i.e., without using an additional memory) and in a modified implementation with storage of a local cost matrix (for nodes of the tree).

For each fixed input length, 10,000 pseudo-random cost matrix generations were performed, for each of which a classic algorithm implementation was started (with recalculation of matrices) and modified algorithm implementation (with matrix storage) [1]. The run-time of the algorithm software implementation, generated number of nodes of search decision tree was measured for each start, and a maximum value of additional memory spent for matrix storage was also measured for modification with matrix storage.

Experiments were carried out on a stationary machine with the following characteristics:

✦ Processor: Intel i7 3770K 3800 GHz;

✦ Random access memory: Kingston KHX-1600C9D3P1 16 GB;

✦ Motherboard: GIGABYTE GA-Z77X-D3H;

✦ Operating system: Fedora 21 workstation.

The algorithms are implemented in language C++. The compiler version: gcc 4.9.2 20150212 (Red Hat 4.9.2-6) (GCC). Let us note that in this regard the total time of computational experiment (10,000 starts for 21 values of the input length) was about 20 hours.

7. Results and discussion

Within the main objective of the study the authors obtained the results shown in *Fig. 2* and *3*. For both options of organization of the decision tree (matrix storage or recomputation) the observed average time grows exponentially. At the same time, we have to note that the option

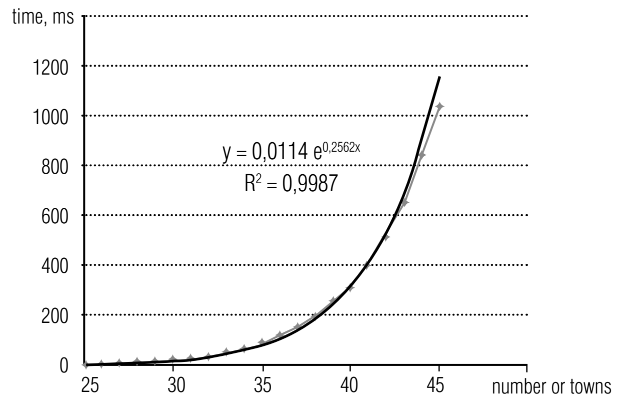

Fig. 2. Dependence of an average time of tour calculation without additional memory on the input length

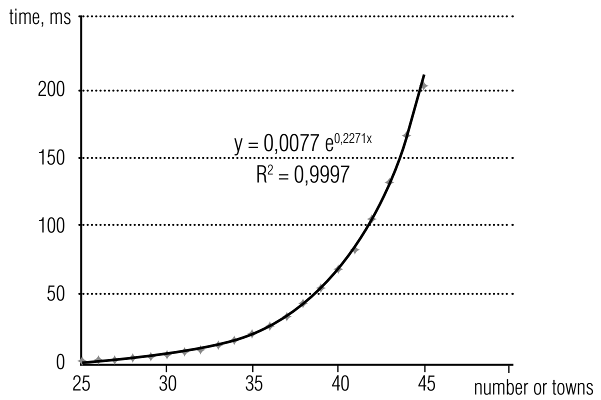

Fig. 3. Dependence of an average time of tour calculation on the input length with the additional memory

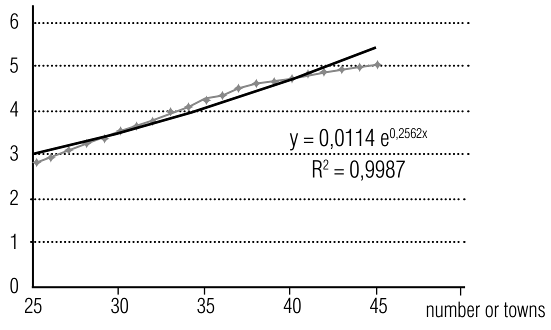

Fig. 4. Dependence of a relation of the average time of tour calculation without additional memory to the average time of tour calculation with the additional memory on input length

with matrix storage results in a change of not only the multiplicative constant, but also the exponent.

From the trend of experimental results (*Fig. 2* and *3*) it follows that with an increase of the input length of the problem the observed reduction of time $d(n)$ (time ratio) will grow in trend $d(n) = 1,478 \cdot e^{0,0291n}$. The results and trend $d(n)$ are provided in *Fig. 4.*

It is evident that the effect of matrix storage will be higher, the deeper the search tree, as a «long» path from the root leads to more complicated recalculations for the original matrix. Our experimental data shows that the use of additional memory reduces the exponent from 0.2562 to 0.2271, i.e. by 0.0291. For the problem with 45 cities this reduction was 5.087 times (from 1.04 to 0,204 sec), all of which indicates considerable option efficiency with additional memory. The resulting trend for $d(n)$ suggests that the observed reduction of time will also increase with the increase of the input length. The resulting prediction is given below.

The following results are obtained for additional goals of the research.

The results on dependence of additional memory capacity on the input length are shown in *Fig. 5.* For input length 45 the additional memory requirements average 30.71 MB, providing a 5-fold reduction of time.

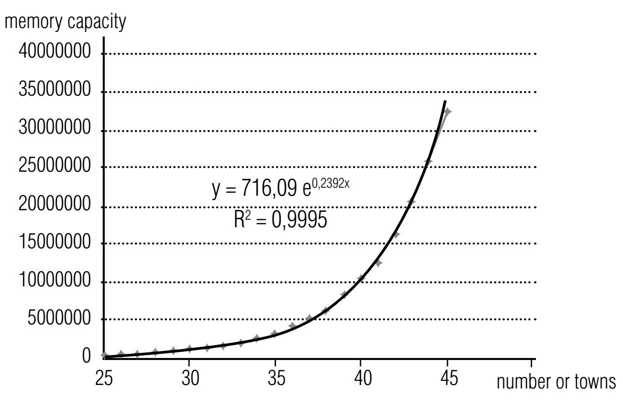

Fig. 5. Dependence of the average capacity of additional memory on the input length

By comparative analysis of the experimental results and data obtained in paper [4] on dependence of the average number of nodes of the search decision tree on the input length, the results obtained by the authors (*Fig. 6*) give the following trend of estimate of an average number of nodes of the decision tree: $R(n) = 5,323 \cdot e^{0,1831n}$ that is qualitatively comparable with the results from [4] – $R(n) = 3,6932 \cdot e^{0,1819n}$. By the exponent, a disagreement with our data is not more than 0.65%.

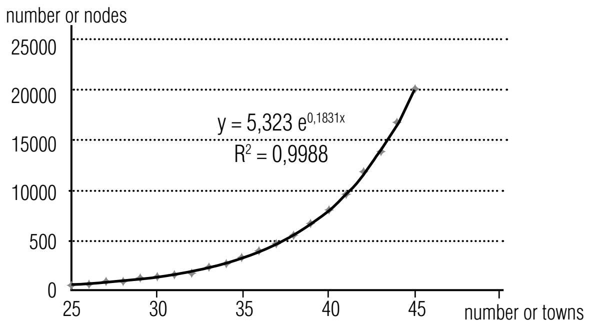

Fig. 6. Dependence of the average number of nodes of the search decision tree on the input length

The discrepancy in the multiplicative constant can be attributed to lower values in a range of studies in paper [4] – from 10 to 30 and may be specifics of the initial data generation. The exponential growth of a number of generated tops of the search decision tree defines the exponential nature of time and capacity characteristics of the software implementations of the method.

Based on the distribution of relative frequencies of the observed time, the following interesting results are obtained which are shown in *Fig. 7.* We have a pronounced left asymmetry of distribution of relative frequencies, indicating that the major part of the observed times is close to the best case. Large times are rare enough. For the point of the sample mean – 204.25 ms a relative frequency of 0.7138 is summarized, and the sample quantile 0.95 is in point 728.70 ms, beyond which only 500 (5%) out of 10,000 observed times lie with a maximum value of 8888.74 ms.

According to the ranges of deviation and sampling root-mean-square deviations for characteristics being studied, the fact that algorithms implementing B&B belong to a class *NPRH* [3] is clearly confirmed by the data on the range of variability of the observed times: if n = 45 for option without matrix storage, we have a range from 3.38 ms to 54058.1 ms at a sample mean of 1039.08 ms. For option with matrix storage from 2.08 ms to 8888.74 ms at sample mean of 205.25 ms. The same significant range is observed for the additional memory spent at n=45 – from 185 936 to 882 836 484 bytes (from 181.57 Kb to 841.39 Mb) with an average value of 30.71 MB. All these ranges

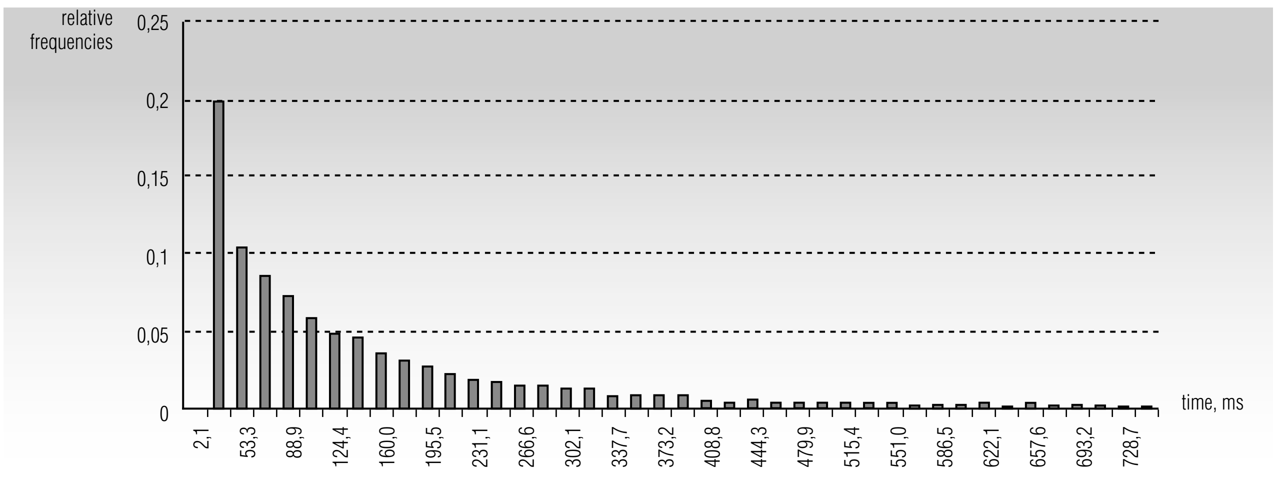

Fig. 7. Relative frequencies of execution times (with matrix storage) for problem with *n*=45 (first 42 half-segments out of 500 by 10,000 experiment results)

are caused by a range of variability of a number of generated nodes of the search tree – from 111 to 820,964 with an average of 20,121.72. The presence of rare but significantly large runs identified both large sample root-mean-square deviations – 1783.65 ms 343.91 ms for the timed and 49 685 928.5 bytes for the spent memory.

Based on the trend of times and additional RAM spent for storing matrices on tops of the search tree, the authors obtained the following prediction of resource characteristics for problems with a high dimensionality presented in *Table 2.*

Table 2.

Prediction of resource characteristics

N-r of cities	Prediction of tour calculation time without additional memory	Prediction of tour calculation time with additional memory	Prediction of time relation	Prediction of average requirements of additional memory
45	1 sec	0.2 sec	5	30.71 MB
54	7 sec	1 sec	7	172.3 MB
70	11.7 min	1 min	11,7	12.47 GB
80	2.5 hours	10 min	15	136.37 GB
88	19.6 hours	1 hour	19,6	924.26 GB
102	29.5 days	1 day	29,5	25.69 TB

Let us note that, beginning from the input length 72, expected memory requirements of the additional memory already exceed the modern standard volume of RAM by 16 GB.

Conclusion

Thus, based on the experimental research consisting of 210,000 solutions of the asymmetric traveling salesman problem with a random uniform generation of cost matrices for a range of input lengths from 25 to 45, and the processing of the results obtained, the article shows that:

✧ both considered options of the classical algorithm for solving the traveling salesman problem using the branch-and-bound method generate software implementations with an exponential execution time dependence on the input length;

✧ the use of additional memory of an acceptable volume leads to a significant reduction of calculating time (up to 5 times with *n*=45, with maximum memory requirements not exceeding 1 GB, with average requirements of 30.71 MB);

✧ the obtained time distribution indicates a pronounced left asymmetry of skewness of distribution, which leads to «small» execution times expected within probabilistic quantile 0.95 in rare but significant (i.e., by orders of magnitudes exceeding an average value) runs;

✧ software implementation of the algorithm with matrix storage can be practically used for the exact solution of the traveling salesman problem with the input length of no more than 70 with the actually available memory of 16 GB with an expected average computation time of about one minute on modern personal computers. ■

References

1. Little J.D.C., Murty K.G., Sweeney D.W., Karel C. (1963) An algorithm for the traveling salesman problem. *Operations Research*, no. 11, pp. 972–989.
2. Goodman S.E., Hedetniemi S.T. (1981) *Vvedenie v razrabotku i analiz algoritmov* [Introduction to the design and analysis of algorithms]. Moscow: Mir (in Russian).

3. Ulyanov M.V. (2008) *Resursno-effektivnye komp'juternye algoritmy. Razrabotka i analiz* [Resource-efficiency computer algorithms. Development and analysis]. Moscow: FIZMATLIT (in Russian).
4. Ermoshin A.S., Plisko V.A. (2006) Issledovanie dereva reshenij metoda vetvej i granic v zadache kommivojazhera [Research of a decision tree in branch-and-bounds method for traveling salesman problem]. P*rogrammnoe i informacionnoe obespechenie sistem razlichnogo naznachenija na baze personal'nyh EVM* [Software and information in different systems based on personal computers]. Moscow: MGAPI, no. 9, pp. 76–82 (in Russian).
5. Sygal I.H., Ivanova A.P. (2007) *Vvedenie v prikladnoe diskretnoe programmirovanie: Modeli i vichislitelnie algoritmi* [Introduction to applied discrete programming: models and algorithms]. Moscow: FIZMATLIT (in Russian).
6. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. (2005) *Algoritmy: postroenie i analiz* [Algorithms: Development and analysis]. Moscow: Williams (in Russian).
7. Virt N. (2010) *Algoritmy i struktury dannyh. Novaja versija dlja Oberona* [Algorithms and data structure. New version for Oberon]. Moscow: DMK Press (in Russian).

РЕСУРСНЫЕ ХАРАКТЕРИСТИКИ СПОСОБОВ ОРГАНИЗАЦИИ ДЕРЕВА РЕШЕНИЙ В МЕТОДЕ ВЕТВЕЙ И ГРАНИЦ ДЛЯ ЗАДАЧИ КОММИВОЯЖЕРА

М.В. УЛЬЯНОВ
доктор технических наук, профессор, ведущий научный сотрудник,
Институт проблем управления им. В.А.Трапезникова РАН;
профессор департамента программной инженерии,
Национальный исследовательский университет «Высшая школа экономики»
Адрес: 117997, г. Москва, ул. Профсоюзная, д. 65
E-mail: muljanov@mail.ru

М.И. ФОМИЧЕВ
студент бакалавриата образовательной программы «Программная инженерия»,
Национальный исследовательский университет «Высшая школа экономики»
Адрес: 101000, г. Москва, ул. Мясницкая, д. 20
E-mail: mikhail.fomichev94@gmail.com

Ресурсная эффективность различных реализаций метода ветвей и границ для классической задачи коммивояжера зависит, в том числе, от способов организации поискового дерева решений, порождаемого этим методом. Классическая дилемма «время-память» реализуется здесь либо вариантом хранения усеченных матриц в вершинах дерева решений, что приводит к сокращению трудоемкости при дополнительных емкостных затратах, либо перевычислением матрицы для текущей вершины, что ведет к увеличению трудоемкости при экономии памяти. Предметом данной статьи является экспериментальное исследование временных характеристик решения задачи коммивояжера методом ветвей и границ с целью определения реального сокращения временных затрат при использовании дополнительной памяти в выбранной структуре хранения дерева решений. Конечной целью исследования является формулировка рекомендаций для реализации метода в практических задачах логистики и бизнес-информатики.

В статье на основе полученных экспериментальных данных показано, что оба рассмотренных варианта классического алгоритма решения задачи коммивояжера методом ветвей и границ порождают программные реализации с экспоненциальной зависимостью времени выполнения от длины входа. Экспериментальные результаты позволяют говорить, что возможность использования дополнительной памяти объемом не более 1 Гб приводит к значительному (до пяти раз) сокращению временных затрат. Прогноз по полученному тренду позволяет сформулировать рекомендацию по практическому применению программной реализации алгоритма метода ветвей и границ с хранением матриц – при реально доступной оперативной памяти в 16 Гб и при ограничении ожидаемого среднего времени счета порядка одной минуты на современных персональных компьютерах могут быть точно решены задачи размерности не более 70.

Ключевые слова: задача коммивояжера, метод ветвей и границ, временная и емкостная эффективность, структуры данных, экспериментальное исследование.

Цитирование: Ulyanov M.V., Fomichev M.I. Resource characteristics of ways to organize a decision tree in the branch-and-bound method for the traveling salesmen problem // Business Informatics. 2015. No. 4 (34). P. 38–46.
DOI: 10.17323/1998-0663.2015.4.38.46.

Литература

1. Little J.D.C., Murty K.G., Sweeney D.W., Karel C. An algorithm for the traveling salesman problem // Operations Research. 1963. No. 11. P. 972–989.
2. Гудман С., Хидетниеми С. Введение в разработку и анализ алгоритмов. М.: Мир, 1981. 368 с.
3. Ульянов М.В. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ. М.: ФИЗМАТЛИТ, 2008. 304 с.
4. Ермошин А.С., Плиско В.А. Исследование дерева решений метода ветвей и границ в задаче коммивояжера // Программное и информационное обеспечение систем различного назначения на базе персональных ЭВМ: Межвузовский сборник научных трудов / Под ред. д.т.н., проф. Б.М.Михайлова. М.: МГАПИ, 2006. Вып. 9. С. 76–82.
5. Сигал И.Х., Иванова А.П. Введение в прикладное дискретное программирование: модели и вычислительные алгоритмы. М.: ФИЗМАТЛИТ, 2007. 304 с.
6. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ. 2-е издание: Пер. с англ. М.: Вильямс, 2005. 1296 с.
7. Вирт Н. Алгоритмы и структуры данных. Новая версия для Оберона / Пер. с англ. М.: ДМК Пресс, 2010. 272 с.