Yugoslav Journal of Operations Research
1 (1991), Number 1, 15–25

AN ALGORITHM FOR M ASYMMETRIC TRAVELLING SALESMAN PROBLEM ON A BANDWIDTH-LIMITED GRAPH

Dragoš CVETKOVIĆ

Faculty of Electrical Engineering, University of Belgrade,
P.O. Box 816, 11001 Belgrade, Yugoslavia

Milan MILOSAVLJEVIĆ

Vladimir DIMITRIJEVIĆ

Institute of Applied Mathematics and Electronics, 11000 Belgrade, Yugoslavia

Abstract. This paper presents a polynomial dynamic programming based algorithm for solving M ($M \geq 1$) travelling salesman problem (TSP) on a directed bandwidth-limited graph, where the number of cities to be visited by each of the M travelling salesmen is specified.

Key words and phrases: travelling salesman problem, directed bandwith-limited graph, dynamic programming, polynomial algorithms

1. INTRODUCTION

Considerable interest has been shown for well-solved cases of *travelling salesman problem* (TSP). According to [8], there are two broad categories of well-solved cases of TSP. In one category are the problems that are special because of restrictions on the matrix C of arc lengths; for example, C may be *upper triangular* or *circulant* matrix. In the second category are the problems in which TSP is to be solved over a network (graph) with particular structure, but with no restriction on the lengths of arcs. In this category is the topic of this paper i.e. TSP on networks having *limited bandwidth*.

Let G be a loopless directed graph (digraph) on vertices $1, 2, \ldots, N$. G is called a bandwidth-limited graph if there exists a positive integer S ($S < N - 1$) such that for any arc (i, j) in G we have $|i - j| \leq S$. The smallest such integer S is called the *bandwidth* of G and is denoted by ω.

To each arc (i, j) of G a *length* (*weight*) c_{ij} is assigned. We define $c_{ij} = +\infty$ if the arc (i, j) does not exist in G. The matrix $C = \|c_{ij}\|_1^N$ is called the *weight matrix* G. The *length* $l(H)$ of a subgraph H of G is defined to be the sum of lengths

of arcs of H. In particular, the length af a path is the sum of lengths of arcs from which it consists. A path consisting from k arcs is called a *k-path*.

We formulate the *M travelling salesmen problem* (M-TSP): *Given a weighted graph G and a positive integer M, find a spanning collection of M disjoint paths (salesmen's tours) with minimal length.*

Variations of this problem include limitations on the number of cities visited by each of the M travelling salesmen. In particular, it is interesting to consider the case when each salesman visits a constant number of cities. In this case, of course, the number of vertices N of G is divisible by the number of salesmen M. Note that the reduction by Belmore and Hong [1] of the M-TSP to the TSP is not applicable to the mentioned variant of the problem.

A TSP is called *symmetric* if the weight matrix is symmetric. Otherwise it is called *asymmetric*.

In [9], [10] (see also [8]) a polynomial algorithm for symmetric TSP on bandwidth-limited graph is described. It is based upon dynamic programming approach. This paper extends the basic ideas of this algorithm for an asymmetric TSP and generalizes it for M ($M \geq 1$) travelling salesmen. For $M > 1$ the numbers of cities to be visited by each of the M travelling salesmen should be specified. Let this specification be provided by a condition α.

We give computational results on VAX/VMS concerning the case when the equal number of cities are to be visited by each of the M travelling salesmen. Our algorithm has been implemented within a programming package called TSP-SOLVER [5].

2. DESCRIPTION OF THE ALGORITHM

Let G be a bandwidth-limited digraph. The algorithm which will be described is related to the case when N, ω, M are independent parametrs with obvious restriction and basically stems from [2].

Let vertices of digraph G be represented as in Fig. 1 for a fixed j ($\omega < j \leq N$).

A_j				B_j				C_j			
∘	∘	⋯	∘	∘	∘	⋯	∘	∘	∘	⋯	∘
1	2	⋯	$j-\omega-1$	$j-\omega$	$j-\omega+1$	⋯	j	$j+1$	$j+2$	⋯	N

Fig. 1

Let us define the sets:

$$A_j = \{1, 2, \ldots, j-\omega-1\},$$
$$B_j = \{j-\omega, j-\omega+1, \ldots, j\},$$
$$C_j = \{j+1, j+2, \ldots, N\}.$$

The algorithm is based on the fact that, due to the bandwidth ω of the digraph G, there are no arcs from vertices of A_j to the vertices of C_j.

Consider M disjoint paths satisfying condition α in digraph G. Let H be a digraph which has the same vertices as G and only arcs from paths mentioned. Let H_j be the subgraph of digraph H induced by the set $A_j \cup B_j$. Digraphs H and H_j contain four types of vertices with respect to the values of vertex indegrees d^- and outdegrees d^+:

1. isolated vertices: $d^- = 0$, $d^+ = 0$ (type 1),
2. starting vertices: $d^- = 0$, $d^+ = 1$ (type 2),
3. terminal vertices: $d^- = 1$, $d^+ = 0$ (type 3),
4. internal vertices: $d^- = 1$, $d^+ = 1$ (type 4).

Degrees of vertices in A_j are the same in H_j and H since in both G and H arcs between A_j, and C_j do not exist.

Let us add vertex $j+1$ to H_j, so that the digraph H_{j+1} is obtained. Vertex $j+1$ is adjacent with 0, 1 or 2 vertices in B_j, Fig. 2. Various possibilities of joining vertex $j+1$ and a vertex $b \in B_j$ that yield a legitimate subgraph H_{j+1}, depend not only on in(out)degree of vertex b, but also on paths in H_j containing vertices in B_j.

The set B_j contains $\omega+1$ elements. Let x_i^j be the i-th vertex in B_j (x_i^j has an absolute position $j-\omega-1+i$ in G). For a vertex $x_i^j \in B_j$, let $u_i^j \in \{1,2,3,4\}$ be its vertex type. Let $t_i^j = (u_i^j, v_i^j, w_i^j)$ be an assigned triple of nonnegative integers defined as follows:

1. if x_i^j is isolated (internal) vertex, then $u_i^j = 1$ ($u_i^j = 4$), and $v_i^j = w_i^j = 0$.
2. if x_i^j is starting (terminal) vertex, the $u_i^j = 2$ ($u_i^j = 3$) is the number of arcs in a path with one endvertex x_i^j, and w_i^j is position (ordinal number) of the other endvertex of the corresponding path. (If the other end of the path is in A_j, we have $w_i^j = 0$).

Triple sequence $T_j = (t_1^j, t_2^j, \ldots, t_{\omega+1}^j)$ descirbes the set B_j, hence the digraph H_j in the extent necessary for futher analysis.

Let $\overline{\mathcal{H}}_j$ be the set of all subgraphs H_j and $\overline{\mathcal{T}}_j$ be the set of corresponding triple sequences T_j. Let us define an equivalence relation on the set $\overline{\mathcal{H}}_j$.

DEFINITION. For a given j, two subgraphs H_j', H_j'' are equivalent ($H_j' \sim H_j''$) if the corresponding triple sequences are equal i.e. $T_j' = T_j''$.

Reduced sets $\mathcal{H}_j \subseteq \overline{\mathcal{H}}_j$, $\mathcal{T}_j \subseteq \overline{\mathcal{T}}_j$ of subgraphs H_j and the coresponding triple sequences T_j respectively, induced by $\sim$ are formed as follows: find the shortest subgraph $H_j \in \overline{\mathcal{H}}_j$ in each equivalence class, include it in $\mathcal{H}_j$, and include at the same time its 'description' i.e. corresponding $T_j \in \mathcal{T}_j$.

The following *algorithm* $\mathcal{A}$ solves the given problem (M-TSP): (All subgraphs H_j are represented by corresponding triple sequences T_j).

1. STEP: *Initialization*: Determine initial set $\mathcal{H}_{\omega+1}$ (see Section 4).

2. STEP: **For** $j = \omega+1, \omega+2, \ldots, N$ **do**:

   Add vertex $j+1$ to every $H_j \in \mathcal{H}_j$ thus obtaining $H_{j+1} \in \overline{\mathcal{H}}_{j+1}$. (For various possibilities of adding vertex $j+1$ that yield legitimate subgraphs H_{j+1} see Section 3).

   Sort triple sequences $T_{j+1} \in \overline{T}_{j+1}$ lexicographicaly. Sets of mutually equal triple sequences correspond to equivalence classes of subgraphs H_{j+1}: Keep the shortest subgraph H_{j+1} in each equivalence class, thus forming reduced sets $\mathcal{T}_{j+1}$ and $\mathcal{H}_{j+1}$.

3. STEP: In set $\mathcal{H}_N$ **find** the subgraph H_N fo the shortest length. *Stop*: Digraph $H = H_N$ is an optimal solution.

Algorithm $\mathcal{A}$ correctly solves the M-TSP problem as stated in the following proposition:

PROPOSITION. *Digraph $H = H_N$ determined in Step 3 of algorithm $\mathcal{A}$ is an optimal solution of the M-TSP problem.*

Proof. Any digraph $F \in \mathcal{H}_N$ has the property that each of its subgraphs F_j belongs to the reduced set of subgraphs $\mathcal{H}_j$. Conversely, all subgraphs K with N vertices containing M paths satisfying condition $K_j \in \mathcal{H}_j$ $(j = \omega+1, \omega+2, \ldots, N)$ are in $\mathcal{H}_N$.

It remains to prove that the digraph H, which is the solution of our problem, has the property $H_j \in \mathcal{H}_j$ $(j = \omega+1, \omega+2, \ldots, N)$. Suppose that for some j we have $H_j \notin \mathcal{H}_j$. This implies that there is a $H'_j \in \mathcal{H}_j$ such that $H'_j \sim H_j$ and $l(H'_j) < l(H_j)$. Then, if H_j is substituted with H'_j the resulting subgraph H' is also solution of the M-TSP with shorter length. This is contradiction with assumption that H is an optimal solution. □

3. GENERATING TRIPLE SEQUENCES

All possibilities of connecting vertex $j+1$ with a vertex in B_j are depicted in Fig. 2.

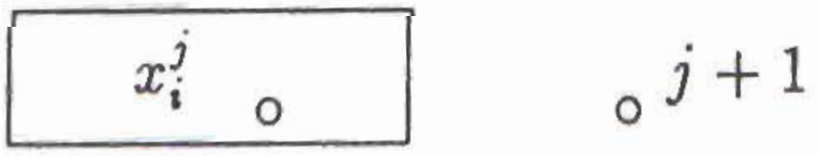

Fig. 2.a

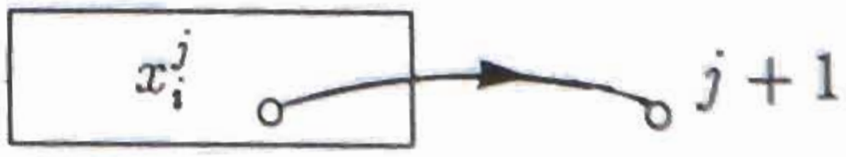

Fig. 2.b

Fig. 2.c

Fig. 2.d

To avoid some technical difficulties in descirptions we shall consider a special condition α: N is divisible by M where $k = N/M - 1$ and each salesman should visit $k+1$ cities.

Let us assume that the reduced set $\mathcal{T}_j$ of triple sequences corresponding to subgraphs $H_j \in \mathcal{H}_j$ is generated.

We generate several triple sequences T_{j+1} from each triple sequence $T_j \in \mathcal{T}_j$ in the following way.

Let $T_j = (t_1^j, t_2^j, \ldots, t_{\omega+1}^j)$ and $T_{j+1} = (t_1^{j+1}, t_2^{j+1}, \ldots t_{\omega+1}^{j+1})$.

For each kind of adding vertex $j+1$ from Fig. 2 we get a sequence T_{j+1}. Triples $t_1^{j+1}, t_2^{j+1}, \ldots, t_\omega^{j+1}$ are obtained from triples $t_2^j, t_3^j, \ldots, t_{\omega+1}^j$ respectively by some modifications depending on the cases in Fig. 2.

Rule A: adding an isolated vertex (see Fig. 2.a.) causes no modification except for decreasing third triple coordinates by 1 if they were greater than zero.

Rules B and C: (connecting vertex by one arc Fig. 2.b and Fig. 2.c) We consider triple t_i^j.

Its first coordinate is changed (vertex type) and other coordinates in the corresponding way. If vertex has become an internal one, other two coordinates become equal 0.

If a vertex x_i^j, represented by triple sequence t_i^j, becomes a starting or terminal vertex, the second coordinate becomes equal to 1 and the third one equal to $\omega + 1$.

Suppose that before modification we had $t_i^j = (u_i^j, v_i^j, w_i^j) = (a, b, c)$ for $2 \leq i \leq \omega + 1$. Then for $c > 0$ in triple t_i^{j+1} we should let second coordinate to be $b+1$, and the third one to be $\omega + 1$.

If we had $b = k$, then this extension is forbiden since H would contain a path of length greater than k.

Rule D: In the final case (connecting the new vertex by two arces as in Fig. 2.d.) only the triples $t_p^j = (a, b, c)$ and $t_q^j = (d, e, f)$ are changed.

First, such an extension is forbiden if there is a path between x_p^j and x_q^j. This case is recognized by conditions $c = q$ and $f = p$. (The extension is also forbiden if at least one of the vertices is internal i.e. $a = 4$ or $d = 4$).

When both vertices are isolated, i.e. $a = 1$ and $d = 1$ there are two possibilities:

(i) $a = 2$ (then $d = 3$), $b = e = 2$, $c = q - 1$, $f = p - 1$;

(ii) $a = 3$ (then $d = 2$), $b = e = 2$, $c = q - 1$, $f = p - 1$.

For $a = 1$ and $d \neq 1, 4$ it follows $a = d$, $b = e + 2$, $c = f - 1$; in this case the extension is forbiden for $e + 2 > k$. If we had $d = 2$, we set new values $d = 4$, $e = 0$, $f = 0$. However, for $f > 0$ before transformation we modify the triple t_f^j; its second coordinate becomes $b + e + 2$ and the third one $p - 1$. The extension is forbidden if $b + e + 2 > k$. We handle the triple t_q^j analogously.

There is a special limitation for the modification of t_2^j. Since this triple becomes t_1^{j+1}, the path which is perhaps attached to vertex x_1^{j+1} must posses the desired number of arcs, i.e. k.

We should describe now the building of triples $t_{\omega+1}^{j+1} = (a, b, c)$ following cases from Fig. 2.

a) $a = 1$, $b = 0$, $c = 0$.

b) Let $t_i^j = (d, e, f)$. Then $a = 3$, $b = e + 1$, $c = f - 1$ for $f > 0$ and $c = 0$ for $f = 0$. The extension is forbidden for $e + 1 > k$.

c) The same as in b) with exlusion $a = 2$.

d) $a = 4$, $b = 0$, $c = 0$.

Generation of new triples can be adapted so that in final solution paths of arbitrary, in advance fixed, lengths exist. In this way the algorithm can be generalized to solve the M-TSP problem with given lengths of solution tours.

4. INITIAL SET CONSTRUCTION

The initial set $\mathcal{H}_{\omega+1}$ is formed in the following way. We introduce auxilary vertices $-\omega, -\omega + 1, \ldots, -1, 0$ into the digraph (see Fig. 3.).

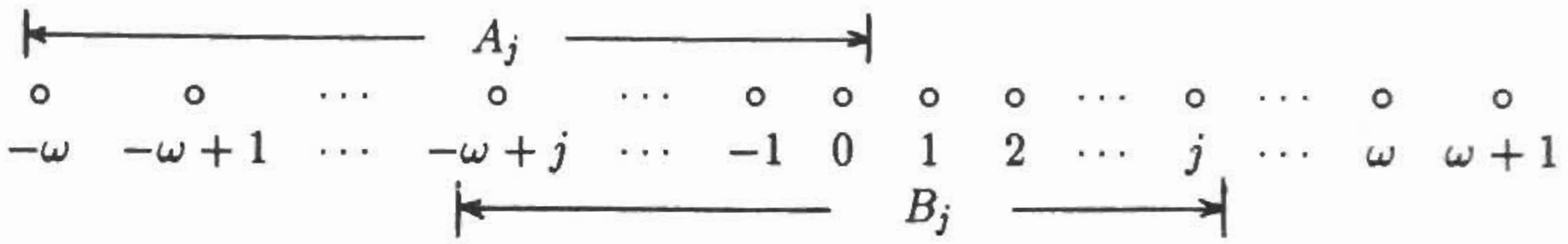

Fig. 3

For $0 \leq j \leq \omega$ let

$$A_j = \{-\omega, -\omega + 1, \ldots, -1, 0\},$$
$$B_j = \{-\omega + j, -\omega + j + 1, \ldots, j\}.$$

We assume that auxilary vertices are isolated in G. Form the set which consist of digraph H_0 induced by vertices $-\omega, -\omega+1, \ldots, -1, 0$. ($H_0$ does not contain any arc). By the procedure described in Section 3 we determine successively sets $\mathcal{H}_j$ ($j = 1, 2, \ldots, \omega + 1$). By passing from $\mathcal{H}_j$ to $\mathcal{H}_{j+1}$, vertex $j + 1$ is not connected to those from $\mathcal{A}_j$ (vertices with a negative label).

5. FINDING THE REPRESENTATIVES OF EQUIVALENCE CLASSES

In this Section we shall describe some detail of the algoritm implementation which have a considerable influence on the efficiency of the algorithm.

In Section 3 we have described how new triples sequences $T_{j+1} \in \mathcal{T}_{j+1}$ are generated from triples $T_j \in \mathcal{T}_j$ using rules A, B, C, D.

In order to reduce mutually equivalent sequences (subgraphs) we sort triple sequences. This sorting is time consuming. The time for sorting can be substantially reduced using a kind of address sorting at the cost of spending more memory space.

Let $u_1, u_2, \ldots, u_{\omega+1}$ be the first triple coordinates from a triple sequence. (The sequence contains $\omega + 1$ triples, where ω is the bandwidth). The first triple coordinates represent vertex type, hence $u_i \in \{1, 2, 3, 4\}$.

The number $s = 4^\omega \cdot u_1 + 4^{\omega-1} \cdot u_2 + \cdots + 4 \cdot u_\omega + u_{\omega+1}$ uniquely determines the sequence $u_1, u_2, \ldots, u_{\omega+1}$.

For $\omega = 10$ we have $4^\omega = 4^{10} = (2^{10})^2 \approx (10^3)^2 = 10^6$ and the number s is contained in the range of INTEGER*4 in FORTRAN.

Generated triple sequences are stored in a triple vector and each sequence gets its ordinal number. When generating a new triple sequence we immediately check whether an equivalent sequence has already been generated. This is achived by the number s since equivalent sequences have the same value for s.

However, non-equivalent sequences can have equal number s. Therefore we introduce a vector EQUI in which, at position s, we put information on already generated triple sequences with the number s.

If EQUI(s) $= 0$, no generated triple sequences has the number s. If such sequences already exist, i.e. EQUI(s) $\neq 0$, their ordinal numbers are chained in another vector, and EQUI(s) contains a pointer to the first triple sequences with the associated number s in the mentioned chained structure.

When generating a new triple sequence we establish in the described way whether an equivalent triple sequence already exists. If it exists, from two eqivalent sequences we selcet the shorter one (in the sense of length of partially constructed salesman's tours) and it is memorized. For each memorized sequence we store into special vectors its length and origin (the sequences on the previous level of sequences from which it is generated; this is useful in reconstructing salesmen's tours at the end of the algorithm).

We shortly mention some other details of the implementation. A triple is internally coded by a unique integer. When generating triple sequence from $\mathcal{T}_{j+1}$ starting from those from $\mathcal{T}_j$ we use records of the later ones whithout copying them into a new memory space.

Consider a triple sequence from $\mathcal{T}_j$ coded by integers $t_1, t_2, \ldots, t_{\omega+1}$ and a pointer u. If $u = p$ we read this sequences as $t_p, t_{p+1}, \ldots, t_{\omega+1}, t_1, t_2, \ldots, t_{p-1}$. Applying Rule A to this sequence we get the sequence whose code is the same with only difference that $u = p + 1$ and $t_p = 0$. Indices addition is performed modulo $\omega + 1$.

Also some addional rules to reduce the number of triple sequences have been implemented, but we shall not describe details here. They are based on the limitation on the number of starting and terminal vertices being ends of paths whose second end is in the set A_j.

6. COMPLEXITY ANALYSIS

We note the following theree observtions:

a) In order to evaluate the complexity of the described algorithm it is essential to find a bound for the cardinality of the set $\mathcal{H}_j$.

An element of the set $\mathcal{T}_j$ is a triple sequence $T_j = (t_1^j, t_2^j, \ldots, t_{\omega+1}^j)$.

The first coordinates belong to the set $\{1, 2, 3, 4\}$. The second triple coordinates yield the number of arcs in paths that begin or terminate in these vertices. The number of arcs in a path is at most k.

The third coordinates belong to the set $\{0, 1, 2, \ldots, \omega + 1\}$.

So, we can conclude that the number of triple sequences is at most $4^{\omega+1} \cdot k^{\omega+1} \cdot (\omega + 2)^{\omega+1}$.

Since $k < N/M$, the number of element of the set $\mathcal{H}_j$ is smaller then $4^{\omega+1} \cdot (N/M)^{\omega+1} \cdot (\omega+2)^{\omega+1}$. Hence, the upper bound is a polynomial $P_{\omega+1}(N)$ of degree $\omega + 1$.

b) When forming $\mathcal{T}_{j+1}$ from a T_j we get a bounded number of sequences T_{j+1}, say at most C. Therefore the total number of generated sequences T_{j+1} is at most $C \cdot P_{\omega+1}(N)$.

Now we have to sort sequences T_j, in order to find equivalence classes of subgraphs H_j. It is known that the address sorting of L objects can be performed with complexity $O(L)$. Hence, the complexity of generating and sorting sequences T_{j+1} is $O(N^{\omega+1})$.

c) Forming $\mathcal{T}_{j+1}$ from $\mathcal{T}_j$ (as in b)) has to be repeated for $j = \omega+2, \omega+3, \ldots, N$ ($N - \omega - 1$ times).

This proves that the described algorithm is polynomial of degree at most $\omega+2$. However, the complexity estimate is not accurate; one should expect much better performance of the algorithm. It seems to be rather difficult to derive a better theoretical complexity estimtion.

Note that standard branch and bound algorithms for TSP remain of an exponential complexity when applied to a bandwidth-limited graph. For example, consider a branch and bound algorithm in which the relaxation task consists in finding a shortest rooted arborescence with s arcs (s-arborescence), where $s = N - M$. When branching we have to "destroy" vertices with outdegrees greater than 1. For the simplicity let $M = 1$ and consider the rooted tree represented in Fig. 4.

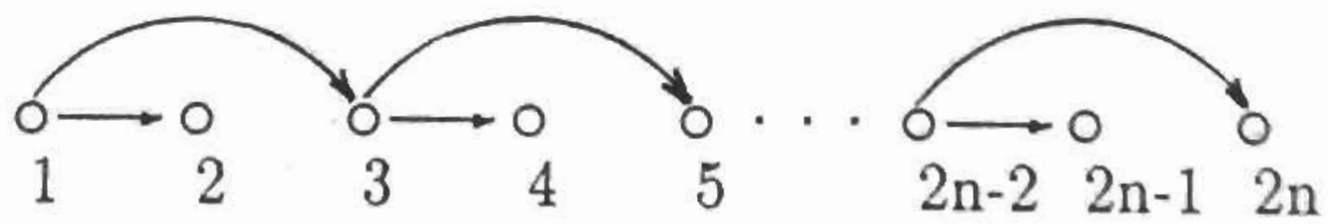

Fig. 4

Suppose this tree is a spanning tree of a bandwidth-limited graph G. Suppose further that arc lengths in this tree are very small when compared to other arc lengths in G. The above branch and bound algorithm would generate 2^n subproblems since we have n vertices of outdegree 2.

However, a disadvantage of our polynomial algorithm when compared with the exponential branch and bound algorithm consists of the fact that it works a constant amount of time for all instances of our problem. Branch and bound

algorithms are sensitive on the distribution of arc lengths so that there are problem instances which are quickly solved while for some "difficult" cases an "exponential" time is necessary.

7. IMPROVING THE BASIC ALGORITHM

In this Section we describe some improvements of our basic algorithm.

Experiments on computer with the above algorithm implemented have shown a bad performance. The number of generated triple sequences is enormous for quite modest values of N and ω. Big memory requirements have implied the usage of virtual memory and this contributed to higher execution times.

Therefore we have developed some procedures for reducing the number of generated triple sequences. The basic idea for these procedures consists in the following.

Some subgraphs H_j (see Section 2) can have so big length that they do not have any chance to be extended to an optimal solutions. Such subgraphs will not be extended any more when putting next vertex into consideration.

Extending notation from Section 2, let G_j and G'_j be the subgraphs of G induced by sets $A_j \cup B_j$ and C_j respectively. Let $H'_j = H - H_j$. Let D be an upper bound for the length of an optimal solution H in G. For example, D can be obtained by a quick heuristic for solving the M-TSP problem on G.

Suppose now d is a lower bound for the length of a shortest subgraph H'_j. The bound d can be obtained by solving the relaxation task on G'_j which should be modified in order to include arcs between B_j and C_j which could appear in H'_j.

If L is the length of H_j, a solution H obtained by extending H_j has the length at least $L + d$. If $L + d > D$, then H_j, cannot be exetended to an optimal solution. Therefore we introduce the quantity $L_c = D - d$ and call it a *critical length*. Obviously one should further develop only those subgraphs H_j whose lengths do not exceeed length L_c.

The described reduction procedure can be applied each time when considering next vertex. Alternatively, the reduction should be performed from time to time considering at once a fixed number of addeed vertices or, adaptively, depending on the number of triple sequences generated.

The heuristic for determining the upper bound D can be organized as follows. Let $\mathcal{F}_j \subset \mathcal{H}_j$ and let us extend only subgraphs H_j from $\mathcal{F}_j$ to the solution H. The length of the best solution H obtained in this way can serve as the bound D. Then go back and extend all triple sequences whose lengths do not exceed the critical length $D - d$, thus obtained.

One can play in many different ways with these reductions. It is also possible to forget some triple sequences whose lengths are smaller than the critical length. In this case our algorithm is transformed into a heuristic since optimal solution could be excluded from consideration.

An extermal case would be to extend just one triple sequences from each set $\mathcal{H}_j$, i.e. one with minimal length. This heuristic is an analogon of the well known *nearest neighbour* heuristic for ordinary TSP.

The rate of reduction of triple sequences at early steps of the algorithm can serve as the basis for defining complexity indices of particular problem instances. Instances with small reduction rate would be classified as "difficult" while those with a big reduction rate as "easy cases". For some details on TSP complexity indices see [6], [7], [3].

8. EXPERIMENTAL RESULTS

Initial experiments have been performed on a few bandwidth-limited graphs with $M = 2$, $N = 20$ and $\omega = 2, 3, 4, 5$ where each of two salesmen should visit 10 cities. The number of equivalence classes of subgraphs H_i $(i \leq j)$ for $j = 4, 5, \ldots, N$, i.e. the number of corresponding triple sequnces, is represented in Table 1. In the empty fields of the table the number is greater than 80000.

$\omega \backslash j$	4	5	6	7	8	9	10	11	12	13
2	41	77	124	190	290	432	624	874	1194	1152
3	68	230	526	968	1593	2517	3881	5841	8543	12617
4	78	371	1481	3801	7663	13563	22246	35532	55328	75321
5	78	419	2180	9852	28004	61366				

$\omega \backslash j$	14	15	16	17	18	19	20
2	2076	2654	3338	4140	5072	6146	7561
3	16905	22969	30595	40051	51633	65669	
4							
5							

Table 1

j	4	5	6	7	8	9	10
without reduction	61	341	1761	7253	19440	40507	73328
with reduction	32	163	776	2558	6644	9669	14671

j	11	12	13	14	15	16
without reduction	122011	192970	300715	457810	678781	980390
with reduction	20398	29304	44021	57067	69952	91976

Table 2

After realizing that it is hopeless to extend all triple sequences we applied the reduction procedure from Section 7 after each 5 vertices. For finding the bound D we used a 3-optimal heuristic and for the bound d we found a shortest forest with suitable parameters [4]. The resulting reduction in the number of generated triple sequences is shown in Table 2. In this example we had $N = 16$, $\omega = 5$, $M = 2$.

Differences in the numbers of triple sequences between Table 1 and Table 2 are due to the fact that in the second experiment additional rules (see the end of

Section 5) for reducing triple sequences have been implemented. Further experiments with different kinds of reduction have indicated that it is possible to achieve reasonable performances of the algorithm.

REFERENCES

[1] M. Bellmore, S. Hong, *Transformation of the multisalesmen problem to the standard travelling salesman problem*, J. Assoc. Comp. Mach. **21** (1974), 500–504.

[2] D. Cvetković, *M travelling salesmen problem in a complete digraph with an asymmetric band weight matrix*, (Serbo-Croatian), unpublished report, Faculty of Elcetrical Engineering, University of Belgrade, 1986, 1–30.

[3] D. Cvetković, *Adaptive solving of the travelling salesman problem*, (Serbo-Croatian), unpublished report, Faculty of Electrical Engineering, University of Belgrade, 1987, 1–48.

[4] D. Cvetković, *Finding shortest rooted forest*, (Serbo-Croatian), unpublished report, Faculty of Electrical Engineering, University of Belgrade, 1986, 1–20.

[5] D. Cvetković, M. Čangalović, V. Dimitrijević, L. Kraus, M. Milosavljević, S. Simić, *TSP-SOLVER-a programming package for the travelling salesman problem*, Univ. Beograd, Publ. Elektroteh. Fak., Ser. Mat. **1** (1990), 41–47.

[6] D. Cvetković, V. Dimitrijević, M. Milosavljević, *Travelling salesman problem complexity indices based on minimal spanning trees*, Graph Theory, Proc. Eighth Yugoslav Symp. Graph Theory, Novi Sad 1987, Univ. Novi Sad, Institute of Mathematics, Novi Sad 1989, 43–51.

[7] V. Dimitrijević, *Adaptive decision in a class of pattern recognition systems*, (Serbo-Croatian), M.Sc. Thesis, Faculty of Electrical Engineering, University of Belgrade, 1986, 1–30.

[8] P. C. Gilmore, E. L. Lawler, D. B. Shmoys, *Well-solved special cases*, in: E. L. Lawler, J. K. Lenstra, A. H. G. Rinnoy Kan, D. B. Shmoys (editors), *The Travelling Salesman Problem*, John Willey & Sons, Chichester-New York-Brisbane-Toronto-Singapore, 1985, 87–143.

[9] B. Monien, I. H. Sudborough, *Bandwidth constrained NP-complete problems*, Proc. 13th Annual ACM Symp. Theory of Computing, 1981, 207–217.

[10] H. D. Ratlif, A. S. Rosenthal, *Order picking in a rectangular warehouse: a solvable case of the travelling salesman problem*, Oper. Res. **31** (1983), 507–521.