PU.M.A. Vol. **17** (2006), No. 3–4, pp. 229–239

Heuristics on a common generalization of TSP and LOP

Z. Blázsik
University of Szeged, Department of Informatics
H–6720 Szeged, Árpád tér 2, Hungary
e-mail: blazsik@inf.u-szeged.hu

and
T. Bartók
University of Szeged, Department of Informatics
H–6720 Szeged, Árpád tér 2, Hungary

and
B. Imreh
University of Szeged, Department of Informatics
H–6720 Szeged, Árpád tér 2, Hungary

and
Cs. Imreh
University of Szeged, Department of Informatics
H–6720 Szeged, Árpád tér 2, Hungary

and
Z. Kovács
University of Szeged, Department of Informatics
H–6720 Szeged, Árpád tér 2, Hungary

(Received: July 12–15, 2006)

Balázs Imreh deceased on 8-th of August 2006, the remaining authors would like to dedicate the paper to his memory.

Abstract. In many important combinatorial optimization problems it is required to find a permutation of vertices of a complete directed graph that minimizes a certain cost function. In this paper we consider a new such optimization model where the objective function is a mixed linear cost function of the functions used in the TSP and the LOP problems. The motivation of this common generalization of TSP and LOP is a practical vehicle routing question. We present and analyse some heuristic algorithms for the solution of the problem.

Mathematics Subject Classifications (2000). 90C27, 90C59

This research has been supported by the Hungarian National Foundation for Scientific Research, Grant T046405 and by the research and development project NKFP-2/015/2004.

1 Introduction

In many important combinatorial optimization problems it is required to find a permutation of vertices of a complete directed graph that minimizes a certain cost function. The most familiar one is the min-cost Hamiltonian path problem – or its closed-path version, the Traveling Salesman Problem (TSP) –, when the cost of a permutation is the sum of the distances of the consecutive node pairs. This problem is one of the most investigated combinatorial problems, an overview of the problem and its variations can be found in [5]. Another problem known as the Linear Ordering Problem (LOP) is to find a linear order of the nodes of a directed graph such that the sum of the arc weights, which are consistent with this order, is as large as possible. An overview on the results concerning the LOP problem can be found in [8], and [9].

In this paper we consider a new optimization model using a mixed linear cost function from these two. The motivation of this common generalization of TSP and LOP is the following practical question. We consider the problem where a vehicle has to visit some places but it can be used for internal transports during its tour. This means that in the case when a place i is visited before a place j then the vehicle can be used to transport some goods from i to j. The profit which can be achieved by such a transport is denoted by B_{ij}. The goal is to find an ordering of the places which maximizes the total profit which can be achieved by the internal transports. This vehicle routing question leads to the LOP model. On the other hand we also have to take into account the cost of the tour. We reduce the profit by this cost and the difference gives the objective function of the mathematical model. We call this problem min-cost Hamiltonian path problem with internal transport, $HPPIT$ in short. We note that there exists another TSP model where the goal is the maximization of the profits, this model is called the selective TSP model (LM90). In the selective TSP version the profit is assigned to the points and the goal is to find the subtour with maximal total profit among the subtours having total distance below a given constant.

HPPIT is a generalization of two NP-hard problems therefore it is also NP-hard. For NP-hard optimization problems, the construction and analysis of heuristic algorithms is a rapidly developing area. By heuristic algorithms we mean fast, (polynomial time) algorithms which do not guarantee an optimal solution in general, but always result in a feasible solution. Heuristic algorithms are important for several reasons. The feasible solutions found by these algorithms can be used in procedures based on branch and bound techniques. Moreover, in practical problems often there is not enough time to find an optimal solution by an exponential algorithm, or the size of the problem is too large to use an exponential algorithm. In these cases, heuristic algorithms can be useful again. It can also occur that we do not need an optimal solution, it is sufficient to find a feasible solution the cost of which is not far from the optimal cost.

In this paper we extend some heuristic algorithms which are defined for the TSP problem to this more general model, and we develop some further

algorithms. The algorithms are analysed by an empirical analysis.

The paper is organized as follows. In Section 2, we define the mathematical model of the problem and we present the used notations. Then, in Section 3, some heuristic algorithms are defined for the solution of the problem. In Section 4 the empirical analysis is presented which is used to compare the presented algorithms.

2 Notions and notation

The vehicle routing problem with inner transportation leads to the following mathematical model. Let $G(V, A)$ be a directed complete graph, where $V = v_0, v_1, \ldots, v_n$ is the set of vertices, (v_0 is the depot, and the other vertices are the places which should be visited by the vehicle). Furthermore two $(n+1)\times(n+1)$ size nonnegative matrices are given, B and D. B_{ij} is the possible profit which can be achieved by the inner transportation from v_i to v_j if v_i is visited before v_j ($B_{ii} = 0$ for each i). D_{ij} gives the cost of travelling from v_i to v_j ($D_{ii} = 0$ for each i).

In the HPPIT problem we would like to find a tour which visits each city exactly once and starts at the depot and returns there at the end of the tour. The objective is to maximize the total profit achieved by the inner transportation taking into account the cost of the tour. A feasible solution can be defined as a permutation p of the set $\{1, \ldots, n\}$. The permutation describes the tour where the vehicle starts and ends at v_0 and visits the other vertices in the order $v_{p(1)}, v_{p(2)}, \ldots, v_{p(n)}$. Then the objective function is given by the formula

$$z(p) = \sum_{0<i<j<n+1} B_{p(i),p(j)} + \sum_{i=1}^{n}(B_{0,p(i)} + B_{p(i),0}) - \sum_{0\leq i<n} D_{p(i),p(i+1)} - D_{p(n),0},$$

and the goal is to maximize this function. In the objective function the profit of the inner transportation from and to v_0 is independent on the order of the other vertices, thus it is a constant.

We can also represent the solutions as the directed cycles of the graph V which contain all of the vertices. Some of the presented algorithms use this representation. In this case a directed cycle containing less vertices than $n + 1$ is called a subtour. For an arbitrary subtour which contains v_0 and for a vertex v of the subtour $PRE(v)$ contains the vertices which are on the path (v_0, v) in the subtour and $SUC(v)$ contains the vertices which are on the path (v, v_0) in the subtour.

3 Heuristic algorithms

In this section we present some heuristic algorithms for the solution of the problem. First we give six tour building algorithms which use different heuristic rules to build a feasible solution. Then a tour improvement algorithm is presented which is based on the neighborhood search technique.

3.1 Tour building techniques

Algorithm TB1 (Tour Building 1)

In this greedy algorithm we define the order of the vertices one by one. In each step we select the vertex which yields the maximal profit, taking into account the travelling cost. (We note that in the iteration steps we do not consider the cost between the inserted vertex and the depot, since in the next iteration this cost will be eliminated.) The algorithm can be defined as follows:

Step 1: Let $0 < k < n+1$ be the value, where $\sum_{0<j<n+1} B_{kj} - D_{0k} = \max_{0<i<n+1} \sum_{0<j<n+1} B_{ij} - D_{0i}$. Let $p(1) = k$. If more than one k exists with this property, then we choose the smallest one. Let $t = 1$. Go to Step 2.

Step 2: If $t = n$, then the procedure is finished, p is defined, the resulted tour is $v_0, v_{p(1)}, \ldots, v_{p(n)}, v_0$. If $t < n$, then let $p(t)$ be the smallest k ($k \neq p(s)$, $s < t$) such that: $-D_{p(t-1),k} + \sum_{0<j<n+1,\ j\neq p(s),\, s<t} B_{kj} = \max_{0<i<n+1,\, i\neq p(s),\, s<t}\{-D_{p(t-1),i} + \sum_{0<j<n+1,\, j\neq q(s),\, s<t} B_{ij}\}$. Increase the value of t and go to Step 2.

Algorithm TB2

In this greedy algorithm we build the order from two directions from forward and from backward. In each step we choose one vertex from backward and one from forward to extend the current partial order. We always choose the vertices which yield the maximal profit (taking into account the travelling cost). The algorithm can be defined as follows.

Step 1: (Definition of the last and the first vertices $v_{p(n)}$ and $v_{p(1)}$): Let $0 < k < n+1$ be the value, where $\sum_{0<j<n+1} B_{jk} - D_{k0} = \max_{0<i<n+1} \sum_{0<j<n+1} B_{ji} - D_{i0}$. If more than one k exists with this property, then we choose the largest one. Let $p(n) = k$, $F = \{k\}$ (F is the set of the ordered vertices). If $n > 1$, then let $0 < k < n+1$, $k \notin F$ be the value, where $\sum_{0<j<n+1,\, j\neq p(n)} B_{kj} - D_{0k} = \max_{0<i<n+1,\, i\notin F} \sum_{0<j<n+1,\, j\neq p(n)} B_{ij} - D_{0i}$. Let $p(1) = k$. If more than one k exists with this property, then we choose the largest one. Let $t = n-1$, $z = 2$, and $F = F \cup \{k\}$. Go to Step 2.

Step 2: In this Step we determine $p(t)$, the next element from backward in the tour. If $z = n$, then the procedure is finished, p is defined, the tour is $v_0, v_{p(1)}, \ldots, v_{p(n)}, v_0$. If $z < n$, then let $p(t)$ be the maximal k, $k \notin F$ such that: $-D_{k,p(t+1)} + \sum_{0<j<n+1,\ j\notin F} B_{jk} = \max_{0<i<n+1,\, i\notin F}\{-D_{i,p(t+1)} + \sum_{0<j<n+1,\, j\notin F} B_{ji}\}$.
Let $z = z+1$, $t = n-t+1$, $F = F \cup \{k\}$ go to Step 3.

Step 3: In this Step we determine $p(t)$, the next element from forward in the tour. If $z = n$, then the procedure is finished, p is defined, the tour is

$v_0, v_{p(1)}, \ldots, v_{p(n)}, v_0$. If $z < n$, then let $p(t)$ be the minimal k, $k \notin F$, such that: $-D_{p(t-1),k} + \sum_{0<j<n+1,\, j\notin F} B_{kj} = \max_{0<i<n+1,\, i\notin F}\{-D_{p(t-1),i} + \sum_{0<j<n+1,\, j\notin F} B_{ij}\}$.

Let $z = z + 1$, $t = n - t$, $F = F \cup \{k\}$ go to Step 2.

Algorithm TB3 (Next insertion)

Initialization Let $r = 0$, $I_r = 0$, $E_r = \{(0, 0)\}$. (v_0 is the starting subtour). Let $r = 0$ and go to the iteration part.

Iteration part (r-th iteration) If $r = n$, then the procedure is ended, the tour given by the edges of E_r is the solution resulted by the algorithm. Otherwise, determine for r the edge (u, v) from the set E_r, where

$$\sum_{i\in PRE(v)} B_{ir} + \sum_{j\in SUC(u)} B_{rj} - D_{ur} - D_{rv} + D_{uv}$$
$$= \max_{(s,t)\in E_r} \left\{ \sum_{i\in PRE(t)} B_{ir} + \sum_{j\in SUC(s)} B_{rj} - D_{sr} - D_{rt} + D_{st} \right\}.$$

Let $E_{r+1} = E_r \setminus \{(u, v)\} \cup \{(u, r), (r, v)\}$ where (u, v) is the selected edge. Increase r by 1, and go to the next iteration.

Algorithm TB4 (Best insertion)

In this algorithm in each step we have a subtour. Then we choose the vertex which can be inserted with the smaller cost into this tour. The algorithm is the extension of the cheapest insertion TSP heuristic algorithm which is analysed in [4] and [3]. The algorithm can be given as follows.

Initialization Let $r = 0$, $I_r = 0$, $E_r = \{(0, 0)\}$ (v_0 is the starting tour). Go to the iteration part.

Iteration part (r-th iteration) If $r = n$, then the procedure is ended, the tour given by the edges of E_r is the solution resulted by the algorithm. Otherwise, determine for each k from set $N \setminus I_r$ the edge $k(u, v)$ from the set E_r, where

$$C(k(u, v)) = \sum_{i\in PRE(v)} B_{ik} + \sum_{j\in SUC(u)} B_{kj} - D_{uk} - D_{kv} + D_{uv}$$
$$= \max_{(s,t)\in E_r} \left\{ \sum_{i\in PRE(t)} B_{ik} + \sum_{j\in SUC(s)} B_{kj} - D_{sk} - D_{kt} + D_{st} \right\}.$$

Let k be the value where the above maximum is maximal. Let $I_{r+1} = I_r \cup k$ and $E_{r+1} = E_r \setminus \{(u, v)\} \cup \{(u, k), (k, v)\}$ where (u, v) is the edge which gives the maximal $C(k(u, v))$ value for the selected k. Increase r by 1, and go to the next iteration.

Algorithm TB5 (Best insertion II)

The algorithm is very similar to algorithm TB4, the main difference is the selection of the starting subtour. In this version we use the following initialization part:

Initialization Choose the pair (i, j) for which the value $B_{ij} - D_{ij}$ is maximal and let $I_r = \{i, j\}$, $E_r = \{(i, j)\}$. (v_i, v_j is the starting subtour). Let $r = 1$, go to the iteration part.

Iteration part The iteration part is the same as in the case of TB4, the only difference that i is considered as the depot during this part.

Algorithm TB6 (Path patching algorithm)

This algorithm uses similar ideas as the two patching algorithm analysed by Karp ([6]). In each step we have a set of paths and we concatenate two of them. We use the following function of the paths, $s(Q)$ and $f(Q)$ are the starting and the final points of the path, function $l(Q)$ is the length of the path, $c(Q)$ is an estimation on the cost of the path. If Q and R are two paths, then QR denotes the paths received by writing the points of R to the end of Q and $\text{con}\,(QR) = c(Q) + c(R) + \sum_{i \in Q,\, j \in R} B_{ij} - D_{f(Q),s(R)}$. The algorithm can be defined as follows.

Initialization Let $L = \{(v_0), (v_1), \ldots, (v_n)\}$ be a list containing paths, let $l(v_i) = 1$, $c(v_i) = 0$ for each i. Go to the iteration part.

Iteration Part

Step 1: If L contains only one path then connect the endpoint and the starting point of the path and the resulted tour is the feasible solution given by the algorithm. If L contains more elements, then go to Step 2.

Step 2: Let $Q \in L$ be the shortest path in L (the element with the minimal $l(Q)$ value). If there are more elements with this property then we choose the one which is the first among them in list L. Delete Q from L. Now let R be the shortest path in L, if there are more elements with this property we choose such R where the value $\max\{\text{con}\,(QR), \text{con}\,(RQ)\}$ is maximal. Delete R from L. If $\text{con}\,(QR) \geq \text{con}\,(RQ)$, then go to Step 3, otherwise go to Step 4.

Step 3: Let $l(QR) = l(Q) + l(R)$, $c(QR) = \text{con}\,(QR)$ and put the path QR to the end of list L. Go to Step 1.

Step 4: Let $l(RQ) = l(Q) + l(R)$, $c(RQ) = \text{con}\,(RQ)$ and put the path RQ to the end of list L. Go to Step 1.

3.2 Tour improvement procedure

Some procedure can be used to improve the solution given by a tour building heuristic. We present here an algorithm which belongs to the class of the neighborhood search algorithms (see [1] for details). We call a tour the neighbor of the ordering X if we can receive it by changing the position of two vertices. This neighborhood definition is also used in the area of single machine scheduling problems (cf. [2]). Now we can define the tour improvement algorithm.

Tour improvement algorithm

Initialization Part Define a tour X by some heuristic procedure. Let $X_0 = X$, $r = 0$, and go to the iteration part.

Iteration Part (r-th iteration)

Step 1: Generate the neighbors of X_r. If $z(X_r) \leq z(X)$ is valid for each neighbor X, then the procedure terminates, X_r is the tour resulted by the algorithm. Otherwise go to Step 2.

Step 2: Let X be the tour among the neighbors of X_r with the maximal objective function value. Let $X_{r+1} = X$, increase the value of r by 1, and go to the next iteration, to Step 1.

Concerning the time complexity of this algorithm we have to note that it can have exponential running time. The size of the neighborhood of a feasible solution is bounded by $O(n^2)$, but there is no polynomial bound on the number of iterations.

4 Empirical analysis

We have implemented the algorithms presented above and analysed their behavior on randomly generated test cases. The test have been performed on a computer which has the following parameters: Intel P4 2.8GHz Prescott, ASUS P4P800SE, 1GB DDR400, Windows XP Pro SP2.

During the empirical analysis we investigated the following 14 algorithms.

- the tour building algorithms $TB1, \ldots, TB6$ defined in Section 3,
- 6 further algorithms $IM1, \ldots, IM6$ which are the tour improvement algorithms using the results of $TB1, \ldots, TB6$ as starting solution,
- BTB which gives the best solution among the solutions of the algorithms TBi,
- *Best* which gives the best solution among the solutions of the algorithms IMi.

To test the algorithms we considered randomly generated matrix pairs of sizes n. We used the following distributions.

- Test A: Both matrices (B and D) are generated uniformly from the interval (0, 500).
- Test B: Matrix B is generated uniformly from the interval (0, 500), matrix D is generated uniformly from the interval (0, $500n/2$).
- Test C: Both matrices (B and D) are generated uniformly from the interval (400, 600).
- Test D: Matrix B is generated uniformly from the interval (400, 600), matrix D is generated uniformly from the interval ($200n$, $300n$)

Tests A and C are used to test the situations where the two matrices have the same distribution. In tests B and D we try to balance the cost and the profit. The cost is the sum of n distances, the total profit is the sum of approximately $n^2/2$ profit values.

We generated 500 matrices of size 100 from each test case. We have performed the algorithms on the generated matrices, the average values of the objective function are summarized in Table 1 and 2.

Table 1: The average profit (TB size 100)

	TB1	TB2	TB3	TB4	TB5	TB6	BTB
Case A	1246680	1235690	1328640	1334750	1326960	1270270	1360430
Case B	1086660	1092810	1087470	1111690	1129850	1064050	1141340
Case C	2399830	2357190	2510870	2513650	2549850	2488090	2563150
Case D	376879	379556	455414	465313	535639	445394	535772

Table 2: The average profit (IM size 100)

	IM1	IM2	IM3	IM4	IM5	IM6	BEST
Case A	1286160	1265300	1337620	1339360	1360090	1337020	1364520
Case B	1111970	1111780	1097645	1118600	1135880	1116050	1146720
Case C	2415320	2369900	2514560	2515520	2563500	2514040	2565210
Case D	386664	387159	458269	467020	537720	467339	537843

We have also considered the running time of the algorithms, the Tour Building algorithms had very small running time, only a few second for the 500 tests. The tour improving algorithms required significantly larger time, we give the data in Table 3 below.

Table 3: The average running time (IM $n = 100$)

	IM1	IM2	IM3	IM4	IM5	IM6	BEST
Case A	103	188	49	66	188	243	539
Case B	9	18	9	22	24	32	67
Case C	99	96	51	69	178	231	935
Case D	10	9	9	24	24	31	119

We also generated matrices of size 1000 from each test case. We generated 200 matrix pairs and performed the algorithms. The tour improvement algorithms have extremely long running time on the test cases A and C (it took more than 4 days to perform 10 tests), therefore we tested these algorithms only on the test cases B and D. The results on the average values of the objective function are summarized in Table 4, the running times are summarized in Table 5.

Table 4: The average profit (size 1000)

	TB1	TB2	TB3	TB4	TB5	TB6	BTB
Case A	125685000	125837000	127493000	127702000	127904000	124903000	127955000
Case B	122797000	122904000	119264000	120317000	120284000	120858000	123046000
Case C	249276000	248938000	250793000	250880000	250970000	249758000	251380000
Case D	48513900	48571600	47911300	48332500	49038800	4858880	49039800
	IM1	IM2	IM3	IM4	IM5	IM6	BEST
Case B	12320800	123278000	119326000	120359000	120330000	121631000	123342000
Case D	48671100	48703200	47923900	48338600	49060500	48860900	49039300

Table 5: The running time IM size 1000 in s

	IM1	IM2	IM3	IM4	IM5	IM6	BEST
Case B	16752	28892	25528	47400	43920	91628	255644
Case D	31948	18052	42092	45376	24292	86244	298730

Besides the average values of the cost function we also collected how many times the best solution was given by the different tour building algorithms. These results are summarized in Table 6.

Table 6: The number of best solutions

	TB1	TB2	TB3	TB4	TB5	TB6
Case A ($n = 100$)	0	0	0	487	13	0
Case B ($n = 100$)	1	15	0	309	170	5
Case C ($n = 100$)	0	0	0	467	23	10
Case D ($n = 100$)	0	0	0	0	500	0
Case A ($n = 1000$)	0	0	0	198	2	0
Case B ($n = 1000$)	0	24	0	172	0	24
Case C ($n = 1000$)	0	0	0	195	3	2
Case D ($n = 1000$)	0	0	0	0	199	1

Evaluation of the results

First consider the objective value of the tour building algorithms on our tests. We can conclude that the best results are given by TB4 and TB5. It is interesting that the performance depends on the test case. For the test cases D algorithm TB5 gives the better results. In the other cases the two algorithm gives similar results, TB4 gives the best solution in most cases but in the average value of the cost function TB5 gives better results for Case B and C. Algorithms TB3 and TB6 give the second best results, their performance is similar, but it is worth noting that TB3 never resulted the best tour building solution. The weakest results are given by TB1 and TB2. TB2 has slightly better performance, in some cases it gives the best tour building solution. We can observe that there is a significant difference (6,8 percentage) between the performances of the best and the worst algorithms. If we consider BTB then we can conclude that it gives only 2,3 percentage improvement compared to the algorithms TB4 and TB5. Similar statement is true for the tour improvement algorithms, they could not achieve much better solution than the tour building algorithms. We think so that the reason is that the objective values achieved by TB4 and TB5 are not far from the optimum. We can also observe that the tour improvement algorithm has more effects on the worst solutions.

Considering the running time of the algorithms we can make the following observations. The tour building algorithms are fast, they use only a few seconds for the tests of size 100. For the larger matrices their running time is a little bit larger and there are significant differences (we did not present the detailed data in the paper). We can conclude, that TB3, TB6 are the fastest algorithms, then TB1 and TB2 follows and TB4 and TB5 are the slowest ones. Thus we can observe that the slowest tour building algorithms gave the best results in the objective function, but not the fastest ones are the worst. There were no significant differences in the running time of these algorithms on the different test cases. Considering the running time of the tour improving algorithms it is an interesting observation that it strongly depends on the test cases. For the

test cases B and D they had smaller running time, on the tests A and C the running time is much larger. This property can be observed on the smaller size inputs, and in the case of the larger size inputs the difference is much higher. The reason may be that the objective function values are larger in the cases A and C and this may cause the difference in the number of iterations. We could also observe that in the cases A and C the tour improvement algorithm could make a slightly larger improvement in the small size instances than in the case B and D, it is likely that this is also true for the larger instances.

References

[1] E. Aarts and J.K. Lenstra, (eds). *Local Search in Combinatorial Optimization*, Wiley Interscience, Chichester, England, 1997.

[2] E.J. Anderson, C.A. Glass and C.N. Potts, *Machine scheduling*, In: Local Search in Combinatorial Optimization, E. Aarts, J.K. Lenstra (eds)., Wiley Interscience, Chichester, England, 1997, 361–414.

[3] A.M. Friese, G. Galbiati and F. Maffioli, *On the worst-case performance of some algorithms for the asymmetric traveling salesman problem*, Networks, **12** (1982), 23–39.

[4] B. Golden, L. Bodin, T. Doyle and W. Stewart Jr., *Approximate traveling salesman algorithms*, Operations Research, **28** (1980), 694–771.

[5] G. Gutin and A.P. Punnen, editors, *The traveling salesman problem and its variations*, Kluwer Academic Publisher, Dordrecht, 2002.

[6] R.M. Karp, *A patching algorithm for the nonsymetric traveling salesman problem*, SIAM J. Comput., **8** (1979), 561–563.

[7] G. Laporte and S. Martello, *The selective travelling salesman problem*, Discrete Applied Mathematics, **26** (1990), 193–207.

[8] G. Reinelt, *The linear ordering problem: algorithms and applications*, Research and Exposition in Mathematics, **8**, Heldermann Verlag, Berlin, 1985.

[9] T. Schiavinotto and T. Stützle, *The linear ordering problem: instances, search space analysis and algorithms*, J. Math. Model. Algorithms, **3** (4) (2004), 367–402.