International Journal of Systems Science
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tsys20

Approximate solutions for the maximum benefit chinese postman problem

W. L. Pearn [a] & W. C. Chiu [b]
[a] Department of Industrial Engineering & Management , National Chiao Tung University , 1001 Tahsueh Rd, Hsinchu, ROC, Taiwan
[b] Institute of Industrial Engineering, National Tsing Hua University , 101, Section 2, Kuang Fu Road, Hsinchu, ROC, Taiwan
Published online: 17 Feb 2007.

To cite this article: W. L. Pearn & W. C. Chiu (2005) Approximate solutions for the maximum benefit chinese postman problem, International Journal of Systems Science, 36:13, 815-822, DOI: 10.1080/00207720500282292

To link to this article: http://dx.doi.org/10.1080/00207720500282292

International Journal of Systems Science
Vol. 36, No. 13, 20 October 2005, 815–822

Approximate solutions for the Maximum Benefit Chinese Postman Problem

W. L. PEARN*† and W. C. CHIU‡

†Department of Industrial Engineering & Management,
National Chiao Tung University, 1001 Tahsueh Rd, Hsinchu, Taiwan, ROC
‡Institute of Industrial Engineering, National Tsing Hua University,
101, Section 2, Kuang Fu Road, Hsinchu, Taiwan, ROC

(*Received 7 August 2003; in final form 8 July 2005*)

The Maximum Benefit Chinese Postman Problem (MBCPP) is an interesting and practical generalization of the classical Chinese Postman Problem, which has many real-world applications. Each arc on the MBCPP network is associated with a service cost for the traversal with service, a deadhead cost for the traversal with no service, and a set of benefits. Each time an arc is traversed, a benefit is generated. The objective of the MBCPP is to find a postman tour traversing a selected set of arcs with the total net benefit maximized. Such a generalization reflects real-world situations more closely. The MBCPP has been shown to be more complicated than the Rural Postman Problem, which is an NP-hard problem. Therefore, it is difficult to find polynomial-time bounded algorithms to solve the problem exactly. In this paper, we first review an existing exact solution procedure, and introduce several heuristic algorithms including the Branch-Scan algorithm, the Connection algorithm with various connection strategies, and the Directed Tree algorithm, to solve the MBCPP approximately. We also apply one-opt, two-opt, component exchange, and component-drop procedures to improve the solutions. The proposed algorithms are tested and compared. Extensive computational results are provided and analysed.

Keywords: Approximate solutions; Maximum Benefit Chinese Postman Problem; Algorithms

1. Introduction

There are numerous generalizations of the well-known Chinese Postman Problem. Examples include the Rural Postman Problem (RPP; Christofides *et al.* 1981, Eiselt *et al.* 1995b, Pearn and Wu 1995), the Hierarchical Postman Problem (HPP; Dror *et al.* 1987, Ghiani and Improta 2000), the *k*-person Chinese Postman Problem (Pearn 1994), the Capacitated Arc Routing Problem (Pearn *et al.* 1987, Assad *et al.* 1987, Golden and Wong 1981, Pearn 1988, 1989, 1991, Ghiani and Improta 2000), Mixed Postman Problem (Pearn and Liu 1995, Pearn and Chou 1999), Windy Postman Problem (Pearn and Li 1994) and many others. The Maximum Benefit Chinese Postman Problem (MBCPP) is another interesting generalization of the Chinese Postman Problem, in which each arc on the network is associated with a service cost for the traversal with service, a deadhead cost for the traversal with no service, and a set of benefits. Each time an arc is traversed a benefit is generated. The objective of the MBCPP is to find a postman tour traversing a selected set of arcs with the total net benefit maximized. Such a generalization reflects real-world situations more closely. Applications directly related to the MBCPP include routing of street sweepers, snowploughs, spraying roads with salt, and inspection of streets for maintenance. These applications have been formulated as traditional arc routing problems in which only one single service traversal on each arc is considered. In real situations, however, the demand on the arc

*Corresponding author. Email: roller@cc.nctu.edu.tw

International Journal of Systems Science
ISSN 0020–7721 print/ISSN 1464–5319 online © 2005 Taylor & Francis
http://www.tandf.co.uk/journals
DOI: 10.1080/00207720500282292

may require multiple service traversals to complete the service. This is true for most snowplough operations, where one single traversal can only remove a portion of the snow covering a street. Multiple service traversals are required, in this case, to complete the removal of the snow. Assad and Golden (1995), Eiselt *et al.* (1995a, b), and Dror (2000) summarized the theory and applications of arc routing problems.

The MBCPP on directed networks was first considered in Malandraki and Daskin (1993) and investigated latter for undirected networks by Pearn and Wang (2003). The problem may be briefly defined as follows. Let $G = (V, A)$ be a directed graph with V representing the set of nodes, and A representing the set of arcs. For each arc $(i,j) \in A$, we are given a non-negative service cost c_{ij}^s for the traversal with service, and a non-negative deadhead cost c_{ij}^d for the traversal with no service, which we expect $c_{ij}^s > c_{ij}^d$. We are also given a set of non-negative benefits $b_{ijr_{ij}}$ from node i to node j for the r_{ij}th traversal, where $r_{ij} = 1, 2, \ldots, n_{ij}$. To reflect real situations, we assume that $b_{ijr_{ij}}$ is non-increasing in r_{ij}. The net cost of the r_{ij}th traversal of the arc (i,j) therefore can be expressed as $c_{ijr_{ij}} = c_{ij}^s - b_{ijr_{ij}}$ for $r_{ij} = 1, 2, \ldots, n_{ij}$, where $n_{ij} = \max\{r_{ij} |\ c_{ijr_{ij}} < c_{ij}^d\}$, and for $r_{ij} \geq n_{ij} + 1$ the net cost is $c_{ijr_{ij}} = c_{ij}^d$. That is, for the traversal of the deadhead arcs, no benefit is generated. Then, the MBCPP is to find a postman tour, starting from the depot, traversing a subset of A, and returning to the same depot with total net cost minimized. We note that the cost and benefit structures given here are more general than those presented in Malandraki and Daskin (1993). In practice, the traversal cost is closely related to the arc distance, and the traversal benefit usually is given depending on the type of applications, which should be provided to the operation office or management personnel.

To illustrate the MBCPP, we consider the following example depicted in figure 1 with four nodes, five arcs, and node 1 as the depot. The arc traversal costs, the deadhead costs, the benefits, and the net costs are shown in table 1. The optimal MBCPP solution is (1, 2, 3, 1, 2, 3, 4, 1) with total service cost 13, total deadhead cost 2, and total benefit 22. Therefore, the total net benefit is 7, which is maximal.

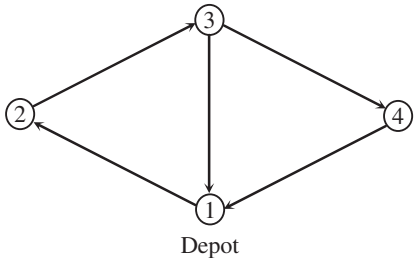

Figure 1. MBCPP example.

2. Existing solution procedures

The MBCPP has been shown to be more complicated than the Rural Postman Problem (which is NP-hard; Christofides *et al.* 1981). In fact, Pearn and Wang (2003) presented a linear transformation to convert the Rural Postman Problem into a special case of the MBCPP. Therefore, it is difficult to find polynomial-time bounded algorithms to solve the problem exactly. Malandraki and Daskin (1993) presented a linear integer programming formulation for the problem, and developed a branch-and-bound type exact algorithm to solve the MBCPP optimally. The exact algorithm is essentially based on a branching rule with some bounding techniques. Subtour elimination constraints are initially removed to relax the original linear-integer programming formulation, which simplifies the MBCPP into the minimal-cost flow problem. The exact algorithm first expands the original network by splitting each arc into several separated parallel arcs with each arc representing a single traversal. The algorithm then relaxes the subtour elimination constraints and solves the resulting relaxed problem as a minimal-cost flow problem. The Minimal-cost flow problem is a polynomial-time solvable network problem, which can be solved efficiently using the polynomial-time bounded algorithms developed in Edmonds and Johnson (1973) on the expanded network. If the solution contains disconnected components, then the minimal-cost flow solution is not feasible to the MBCPP. In this case, the exact algorithm continues and applies a branching rule by adding some bounding constraints to include or exclude certain arcs, solving the relaxed minimal-cost flow problem iteratively to find feasible solutions.

For the MBCPP example depicted in figure 1, the transformed network, expanded from the original MBCPP, is shown in figure 2 with arc net costs indicated. The resulting network after applying the minimal-cost flow algorithm to the expanded network is displayed in figure 3. Since the minimal-cost flow solution contains only one component, the solution is feasible to the MBCPP, which is also optimal in this case. Unfortunately, the computational time required

Table 1. Traversal costs and benefits for the MBCPP example.

Arc	c_{ij}^s	c_{ij}^d	$b_{ijr_{ij}}$	$c_{ijr_{ij}}$
(1, 2)	2	1	3	−1, 1
(2, 3)	2	1	3	−1, 1
(3, 4)	3	2	5, 2	−2, 1, 2
(4, 1)	3	2	5, 2	−2, 1, 2
(3, 1)	3	2	6, 2	−3, 1, 2

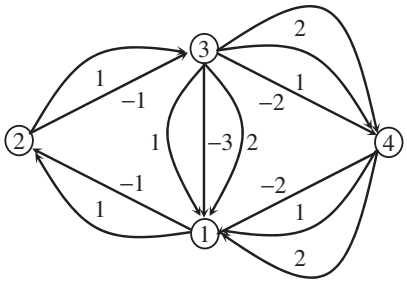

Figure 2. Expanded network.

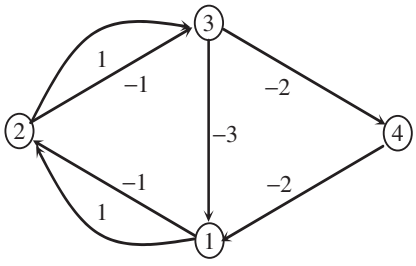

Figure 3. Minimal-cost flow solution over the expanded network.

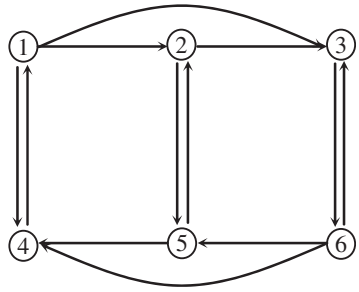

Figure 4. MBCPP example.

Table 2. Arcs and the costs for the MBCPP example depicted in figure 4.

Arc	$c_{ijr_{ij}}$	Arc	$c_{ijr_{ij}}$	Arc	$c_{ijr_{ij}}$
(1, 4)	−10, 1, 10	(3, 6)	−10, 1, 10	(6, 5)	2, 10
(4, 1)	−10, 1, 10	(6, 3)	−10, 1, 10	(5, 4)	2, 10
(2, 5)	−10, 1, 10	(1, 2)	2, 10	(1, 3)	2, 10
(5, 2)	−10, 1, 10	(2, 3)	2, 10	(6, 4)	2, 10

for this exact approach grows exponentially as a function of the size of the given network (total number of nodes $|V|$, and total number of arcs $|A|$). The proposed exact approach, therefore, is computationally inefficient. Only problems of small or moderate size can be solved within a reasonable amount of computer time. In this paper, we take a different approach. We present several heuristic solution procedures to solve the problem approximately. We also consider some procedures to improve the heuristic solutions.

3. Heuristic algorithms

In the following, we introduce several heuristic algorithms including (A) the Branch-Scan algorithm, (B) the Connection algorithm, and (C) the DiTree algorithm to solve the MBCPP approximately. Phase one of these algorithms is the same, which involves finding a symmetric directed graph and solving the corresponding minimal-cost flow problem. We also consider some improvement procedures to improve the solutions. These algorithms are described in the following. It should be noted that if the three algorithms are applied to CPP, then any solutions obtained are optimal.

3.1. *Branch-Scan Algorithm*

The Branch-Scan algorithm is essentially based on the exact algorithm proposed by Malandraki and Daskin (1993). In executing the branch-and-bound procedure, the algorithm simplifies the search procedure using the depth-first strategy. In each level of the branching tree, the Branch-Scan algorithm considers the branch with the lowest bound. Then, the algorithm proceeds downward to the next level of the branching tree, continuing with this depth-first search procedure until the branching tree terminates with the branching (search node is fathomed). Stopping rules with bounds on the maximum computer run time or until the first feasible solution is found are also enforced so that the number of branching and searching iterations would not grow exponentially. If the solution corresponding to the fathomed node is infeasible, the algorithm backtracks the branching tree to the previous level to find an alternative non-fathomed node.

For the MBCPP example depicted in figure 4, which is presented in Malandraki and Daskin (1993) with some modification, the net costs are displayed in table 2. The expanded network contains three copies of the arcs (1, 4), (4, 1), (2, 5), (5, 2), (3, 6), (6, 3), and two copies of the arcs (1, 2), (2, 3), (1, 3), (6, 5), (5, 4), (6, 4). The minimum-cost flow solution over the expanded network consists of three disjoint components: $C_1 = \{1, 4\}$, $C_2 = \{2, 3\}$, and $C_3 = \{3, 6\}$ with each component containing two arcs forming a cycle, which obviously is infeasible.

To illustrate the Branch-Scan algorithm, we first branch on the arc (1, 2) to determine if the component C_2 should be included in the solution. We then choose the node with the lowest bound and continue to branch

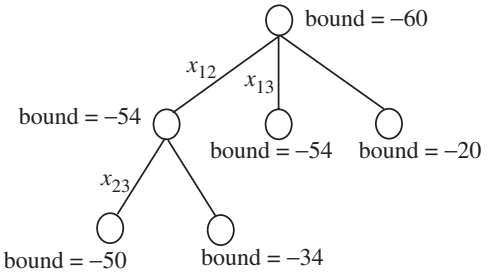

Figure 5. Branching tree generated by the Branch-Scan algorithm.

Table 3. Performance of the Search algorithm on the 15 test problems.

	$\|V\|$	$\|A\|$	Search	Optimal	R
1	10	54	43	43	1.00
2	10	54	51	51	1.00
3	10	81	52	52	1.00
4	10	81	60	60	1.00
5	10	81	27	27	1.00
6	20	228	105	105	1.00
7	20	228	196	196	1.00
8	20	228	97	99	0.98
9	20	342	161	161	1.00
10	20	342	128	128	1.00
11	30	174	113	121	0.93
12	30	522	223	223	1.00
13	30	522	490	490	1.00
14	30	783	317	317	1.00
15	30	783	242	243	1.00

on the arc (2, 3), to determine if the solution should include the component C_3. The resulting branching tree is shown in figure 5 with bounds indicated, where x_{ij} represents the first traversal of the arc (i,j). The solution obtained by applying the Branch-Scan algorithm is -50 with a total net benefit of 50.

We experiment with the Branch-Scan algorithm on 15 sample problems with sizes no greater than $|V| = 30$, and $|A| = 783$, where the maximum CPU time restriction was initially relaxed. The results are displayed in table 3 with $|V|$ representing the number of nodes in G, $|A|$ representing the number of arcs in G, and R representing the ratio of the Search solution and the optimal solution. The results show that the Branch-Scan algorithm works reasonably well. In fact, the Branch-Scan algorithm yields 12 optimal solutions (out of 15). The average ratio of the Search solution over the optimal solution exceeds 0.99. The experiment also shows that the CPU time required is roughly $\alpha|A|$ with $\alpha = 0.02$. Therefore, in practice, we may set $\alpha|A|$ as the maximum run time in CPU seconds.

3.2. *Connection algorithm*

The Connection algorithm consists of two phases. In Phase I, the algorithm first expands the original network by splitting each arc into several separated arcs, with each representing a single traversal, and then relaxes the subtour elimination constraints to solve the remaining problem as a minimal-cost flow problem over the expanded network. If the solution contains only one component, the algorithm terminates, and the solution is optimal. Otherwise, the algorithm proceeds to Phase II executing the linking procedure to construct a cycle by connecting nodes selected from each component. The Connection algorithm considers several strategies for the linking procedure to generate the most profitable connection, which we refer to as the Neighbour Connection (NC), the Savings Connection (SC), and the Insertion Connection (IC), which are described below.

3.2.1. Neighbour Connection (NC). Let $C = \{Cr\}_{r=1}^{k}$ be the set of components generated from phase I. For the Neighbour Connection, the linking procedure starts with node i in a component C_{ro}. Then, the linking procedure selects a component C_r (only one) from $W = C - \{C_{ro}\}$, and connects i in C_{ro} to j in C_r minimizing $l(i,j)$, where $l(i,j)$ is the net cost of the path from i to j. Update $W = W - \{C_r\}$. At each iteration, the linking procedure selects a node in a component from W and links the node to the previous connected components so that the added net cost is minimized. The linking procedure terminates when $W = \phi$. The Neighbour Connection considers three criteria to generate the most profitable connection, which we refer to as the *Forward* (NC_F), *Backward* (NC_B), and *Minimum* (NC_M) criteria. For the *Forward*, p is chosen so that $l(j,p)$ is minimized, where j is the ending point of the previous connected component; for the *Backward*, p is chosen so that $l(p,i)$ is minimized, where i is the starting-point of the initial connected component; for the *Minimum*, p is chosen so that $\min\{l(j,p), l(p,i)\}$ is minimized, where $p \in C_r \in W$. Repeat phase II for all $i \in \{C_r\}_{r=1}^{k}$, with each generating three solutions based on those criteria. The best one of all is chosen as the Neighbour Connection solution.

3.2.2. Savings Connection (SC). The linking procedure starts with node i in a component C_{ro}. Then, the linking procedure selects a node p from each component $C_r \in W = C - \{C_{ro}\}$ forming a set of initial cycles (i,p) (p,i), and computes the savings $s(p,p') = \{l(p,i) + l(i,p') - l(p,p') |\ i \in C_{ro},\ p \in C_r,\ p' \in C_{r'}\}$ for each pair of cycles, and orders them from the largest to the smallest. At each iteration, the linking procedure

chooses the largest $s(p,p')$ and merges two cycles by replacing arcs (p,i), (i,p') by arc (p,p'). The linking procedure terminates when no savings can be made. The Savings Connection considers two criteria for selecting the linking nodes, which we refer to as the *one-way* (SC_I) and *two-way* (SC_{II}) criteria. For the *one-way* criterion, p is chosen from the component $C_{r'}$ so that $\min\{l(i,p),\ l(p,i)\}$ is minimized; for the *two-way* criterion, p is chosen from the component so that $l(i,p)+l(p,i)$ is minimized.

3.2.3. Insertion Connection (IC). The linking procedure starts with node i in the component C_{ro}. Then, the linking procedure selects a node j in C_r from $W=C-\{C_{ro}\}$ forming a cycle i–j–i with minimal $l(i,j)+l(j,i)$. Update $W=C-\{C_r\}$. At each iteration, the linking procedure selects a node in a component from W and inserts the node and the corresponding component to the previous connected components so that the added net cost is minimized. The linking procedure terminates when $W=\phi$. The Insertion Connection considers two criteria to generate the most profitable connection, which we refer to as the *Nearest* (IC_N), and the *Profit* (IC_P) criteria. For the *Nearest*, p is chosen so that $\min\{l(i,p),l(p,i)\}$ is minimized, then the procedure finds the arc (i,j) from the previous cycle maximizing $l(i,j)-l(i,p)-l(p,j)$ and inserts p between i and j. For the *Profit*, p is chosen so that $\{l(i,j)-l(i,p)-l(p,j)+b(p)\}$ is maximized, where $b(p)$ is the total net benefit of the component containing the node p. The best one is chosen as the IC solution.

For further improvement, we consider the following procedures called the *one-opt*, *two-opt*, *component-exchange*, and *component-drop* procedures. Given a MBCPP solution, the *one-opt* procedure considers the connected components and replaces one linking node with another linking node in the same component if it yields a net cost reduction. The *two-opt* procedure replaces two adjacent linking nodes simultaneously with another two linking nodes from the same components if it yields a net cost reduction. The *component-exchange* procedure interchanges two adjacent linking nodes if it yields a net cost reduction. The *component-drop* procedure drops a component from the current solution if it yields a net cost reduction. Repeat the four improvement procedures until no more net cost reduction can be made. Table 4 displays the performance of the three strategies NC, SC, and IC for the Connection algorithm on the 15 test problems in terms of the ratio (R) of the generated solutions over the optimal solutions. The performances of those three strategies are rather close to each other. They all perform quite well for some problems but poorly for others. Although all three strategies are efficient, the Insertion strategy requires more computer time than the other two owing to the extensive number of checkings for finding the appropriate node for most profitable connection.

Table 4. Performance of the three Connection Algorithms, NC, SC, and IC, on the 15 test problems, where $R(\bullet)$ = (heuristic solution)/(optimal solution).

	$\|V\|$	$\|A\|$	R(NC)	R(SC)	R(IC)
1	10	54	0.44	0.44	0.44
2	10	54	1.00	1.00	1.00
3	10	81	0.79	0.79	0.79
4	10	81	0.83	0.83	0.83
5	10	81	0.26	0.26	0.26
6	20	228	1.00	0.88	1.00
7	20	228	0.99	1.00	0.98
8	20	228	1.00	0.97	0.97
9	20	342	1.00	1.00	1.00
10	20	342	1.00	0.99	1.00
11	30	174	0.97	0.97	0.97
12	30	522	1.00	0.98	1.00
13	30	522	1.00	0.99	1.00
14	30	783	1.00	0.99	1.00
15	30	783	1.00	1.00	1.00

3.3. *Ditree expansion algorithm*

The Ditree expansion algorithm consists of two phases. In phase I, the algorithm first expands the original network by splitting each arc into several separated arcs, with each representing a single traversal, and then relaxes the subtour elimination constraints to solve the remaining problem as a minimal-cost flow problem over the expanded network. If the solution contains only one component, the algorithm terminates, and the solution is optimal. Otherwise, the algorithm proceeds to phase II by first defining the net cost between every pair of components as $l(C_r,C_{r'},\lambda)=\min_{x,y}\{l(x,y,\lambda)|\ x\in C_r, y\in C_{r'}\}$, where $l(x,y,\lambda)$ is the cost of the least-cost path between x and y over the modified network with λ (initially set to zero) added to every arc on the paths linking C_r and $C_{r'}$. Then, the algorithm constructs a Shortest Spanning Arborescence (SSA; Christofides *et al.* 1981) to connect all the components. Let T be the set of arcs from the SSA solution. The Ditree expansion algorithm marks the arcs in T as 'required to traverse' and solves the minimal-cost flow problem over the expanded network generating a complete MBCPP solution. Different values of λ may lead different SSAs. In fact, for a large value of λ the SSA solution tends to choose paths containing a smaller number of arcs (and hence a smaller number of λs) in linking the components. Therefore, we can generate various MBCPP solutions

by adding chosen values of λ to the arcs in T, choosing different nodes as the root of SSA, and select the best among all as the solution from this approach. We experiment with the Ditree expansion algorithm on the same 15 test problems, where the parameter λ is initially set to 0, 1, 2, 3, 4, and 5. The preliminary results indicate that the solutions obtained on these problems achieve the best problem solutions for $\lambda = 0, 1$. Therefore, we limit the choice of λ to 0 and 1.

3.4. *Improvement procedures*

We note that the four improvement procedures discussed earlier, the *one-opt*, *two-opt*, *component-exchange*, and *component-drop*, are applied to the Branch-Scan algorithm, the three Connection algorithms (NC, SC, IC), and the Ditree expansion algorithm, in the same order, to improve the solutions generated. We also note that the order of the four improvement procedures may affect the final solutions but not significantly.

4. Computational comparisons

To test and compare the proposed solution procedures, we generate three sets of test problems, which are randomly generated. Test problems in set I have the following network characteristics: the number of nodes, $10 \le |V| \le 30$, the number of arcs, $27 \le |A| \le 435$, the benefit layers are numbered from 1 to 5, and the net costs are set to $-100 \le c_{ijr_{ij}} \le 100$. Test problems in set II have the following network characteristics: the number of nodes, $40 \le |V| \le 100$, the number of arcs, $78 \le |A| \le 4950$, the benefit layers are numbered from 1 to 5, and the net costs are set to $-100 \le c_{ijr_{ij}} \le 100$. For the test problems in set III: the number of nodes, $125 \le |V| \le 200$, the number of arcs, $1550 \le |A| \le 19\,900$, the benefit layers are numbered from 1 to 5, and the net costs are set to $-100 \le c_{ijr_{ij}} \le 100$. Some networks are dense, and others are sparse.

For evaluating the accuracy of the obtained solutions, we also implement the branch-and-bound exact algorithm proposed by Malandraki and Daskin (1993) to find the problem optimal solutions. The optimal solutions can be found only for the small and moderate size of problems in set I (within a reasonable amount of computer time) because of the complexity of the problems. We note that the upper bounds, used as a convenient reference point for assessing the accuracy of the heuristic solutions, are obtained from solving the MBCPP by relaxing the subtour elimination constraints (which is reduced to the minimal-cost flow problem). All the computations and experiments are implemented using FORTRAN programming language, executed on Pentium IV personal computers. The comparison, summarized in tables 5–7, is based on the average percentage below the problem optimal solutions, worse percentage below the optimal solution, number of optimal solutions obtained, number of best solutions obtained, and number of worst percentages obtained. For problems in sets II and III,

Table 5. Performance comparisons of problem set I of the five algorithms (30 problems).

	Branch	NC	SC	IC	Ditree
Average % below the optimal solution	0.71	6.80	9.00	7.00	9.16
Worst % below the optimal solution	6.61	74.07	74.07	74.07	59.09
Number of optimal solutions obtained	16	15	8	13	3
Number of best solutions obtained	22	19	13	16	9
Number of worst solutions obtained	4	3	6	5	15
Average CPU seconds	0.52	1.41	3.34	5.23	0.85

Table 6. Performance comparisons of problem set II of the five algorithms (40 problems).

	Branch	NC	SC	IC	Ditree
Average % below the upper bound	51.68	6.27	7.62	6.12	9.75
Worst % below the upper bound	98.08	83.33	83.33	83.33	47.83
Number of upper bounds obtained	5	9	4	8	2
Number of best solutions obtained	8	18	7	23	6
Number of worst solutions obtained	28	1	1	1	2
Average CPU seconds	2.52	9.41	19.34	28.23	2.12

Table 7. Performance comparisons of problem set III of the five algorithms (30 problems).

	Branch	NC	SC	IC	Ditree
Average % below the upper bound	80.88	7.20	8.80	4.60	13.03
Worst % below the upper bound	99.40	32.26	32.26	32.26	32.26
Number of upper bounds obtained	1	2	2	2	1
Number of best solutions obtained	1	18	3	13	2
Number of worst solutions obtained	28	1	1	1	1
Average CPU seconds	5.52	26.41	49.34	58.23	6.12

Table 8. Average percentage below the optimal (upper bound) for various α.

	Set I	Set II	Set III
$\alpha = 0.02$	0.71	51.68	80.88
$\alpha = 0.05$	0.18	39.68	78.12
$\alpha = 0.10$	0.01	34.08	76.38
$\alpha = 0.20$	0.01	26.25	72.82
$\alpha = 0.50$	0.01	20.33	65.74

upper bounds will be used in the comparisons. In comparing the three algorithms the test results show that:

(1) The Branch-Scan algorithm yields 22 best solutions in problem set I (out of 30), which outperforms that of the other four algorithms. In fact, the Branch-Scan algorithm yields an average of 0.71 percent below the optimal solution, the worst 6.61 percent below the optimal solution, and 22 optimal solutions. However, the Branch-Scan algorithm yields only eight best solutions in problem set II and one best solution in problem set III.
(2) For the Connection algorithm, NC yields 19, SC yields 13, and IC yields 16 best solutions in problem set I. In problem set II, NC yields 18, SC yields seven, and IC yields 23 best solutions. In problem set III, NC yields 18, SC yields three, and IC yields 13 best solutions. The performances of the three strategies seem to be close to each other, but the IC strategy requires more computer run time than the other two. Overall, the ratio of performance seems to be stable for all problem sizes.
(3) The Ditree algorithm yields nine best solutions in problem set I, six best solutions in problem set II, and two best solutions in problem set III. In fact, the Ditree algorithm yields an average of 9.16 percent below the optimal in problem set I, 9.75 percent below the upper bound in problem set II, and 13.03 percent below the upper bound in problem set III.

The Branch-Scan algorithm does not seem to work well, owing to insufficient run time as the problem size increases. For further testing, we set a different α value to increase the run time for the same test problems I, II, and III. Table 8 displays the solutions in term of the average percentage below the optimal, or the upper bounds of the Branch-Scan algorithm, for $\alpha = 0.02$, 0.05, 0.10, 0.20, and 0.50. For $\alpha = 0.02$, the average percentage below the optimal solutions (or the upper bound) is 0.71 percent for problems in set I, 51.68 percent for problems in set II, and 80.88 percent for problems in set III. For $\alpha = 0.50$, the average percentage below the optimal solutions (or the upper bound) decreases to 0.01 percent for problems in set I, 20.33 percent for problems in set II, and 65.74 percent for problems in set III. The results indicate that the performance of the Branch-Scan algorithm varies depending on the network size and structure. Obviously, if more run time is used for the Branch-Scan algorithm, better solutions can be obtained. Thus, for small and moderate problem sizes, the Branch-Scan algorithm is a good solution procedure.

Table 9. Number of optimal solutions for the NC, SC, IC, and Ditree algorithms.

Test problems	Problem characteristic	NC	SC	IC	Ditree
IV-a	$c_{ij1} + 1$	5	5	6	6
IV-b	$c_{ij1} + 2$	9	8	8	8
IV-c	$c_{ij1} + 3$	9	9	9	9
IV-d	$c_{ij1} + 4$	10	10	10	10

Table 10. Average percentage below optimal solution for the NC, SC, IC, and Ditree algorithms.

Test problems	Problem characteristic	NC	SC	IC	Ditree
IV-a	$c_{ij1} + 1$	10.33	10.33	10.67	5.31
IV-b	$c_{ij1} + 2$	3.28	3.38	3.34	3.76
IV-c	$c_{ij1} + 3$	0.16	0.16	0.21	0.05
IV-d	$c_{ij1} + 4$	0.00	0.00	0.00	0.00

Table 11. Worst percentage below the optimal solution for the NC, SC, IC, and Ditree algorithms.

Test problems	Problem characteristic	NC	SC	IC	Ditree
IV-a	$c_{ij1} + 1$	51.35	51.35	51.35	2.54
IV-b	$c_{ij1} + 2$	32.83	32.83	32.83	35.82
IV-c	$c_{ij1} + 3$	1.57	1.57	2.10	0.53
IV-d	$c_{ij1} + 4$	0.00	0.00	0.00	0.00

To further investigate the impact of the cost/benefit structure on the solutions of the four algorithms, we experiment with the sample problems tested earlier by arbitrarily adding the values, 1, 2, 3, and 4, to the net benefit of the first layer on each arc, for 10 sample problems tested earlier. We denote the new problem sets as IV-a, IV-b, IV-c, and IV-d, respectively. The results, as displayed in tables 9–11, indicate that for all four algorithms, the greater net benefit we add to the first layer, the better the solution.

The results in tables 9–11 indicate that the quality of the solutions obtained is highly dependent

on the weight placed on the benefit for the first layer. This is because as the weight of the benefit is placed more on the first layer and less on the other layers, the problem gradually reduces to the classical CPP, and the solutions obtained therefore tend to be optimal. This is expected, since all four algorithms can solve the classical CPP optimally, as stated earlier.

5. Conclusions

The Maximum Benefit Chinese Postman Problem (MBCPP) is a practical generalization of the Chinese Postman Problem, which has many applications. The MBCPP has been shown to be computationally intractable. Therefore, it is difficult to find polynomial-time bounded algorithms to solve the problem exactly. In this paper, we proposed several algorithms to solve the MBCPP approximately, including the Branch-Scan algorithm, Connection algorithm, and Ditree expansion algorithm. We also considered several criteria for the Connection algorithm. Those criteria include the Forward Neighbour, Backward Neighbour, Minimum Neighbour, One-way Saving, Two-way Saving, Nearest Insertion, and Profit Insertion. We experimented with the proposed algorithms on many problems, which were randomly generated. The results indicated that the Branch-Scan algorithm outperformed the other two algorithms for the MBCPP of small or moderate sizes. For a large MBCPP, the Connection algorithm seems to perform best.

Acknowledgements

The authors would like to thank the three anonymous referees for their constructive comments and careful readings, which significantly improved the paper.

References

A. Assad and B. Golden, "Arc routing methods and applications", in *Network Routing Handbooks in OR & MS*, Vol. 8, Amsterdam: Elsevier Science, 1995.

A.A. Assad, W.L. Pearn and B.L. Golden, "The capacitated Chinese postman problem: lower bounds and solvable cases", *Am. J. Math. Manag. Sci.*, 7, pp. 63–88, 1987.

N. Christofides, V. Campos, A. Corberan and E. Mota, "An algorithm for the rural postman problem", in *Imperial College Report IC-OR*, 1981.

M. Dror, *Arc Routing. Theory, Solutions and Applications*. Boston, MA: Kluwer Academic, 2000.

M. Dror, H. Stern and P. Trudeau, "Postman tour on a graph with precedence relation on arcs", *Networks*, 17, pp. 283–294, 1987.

J. Edmonds and E.L. Johnson, "Matching, Euler tours and the Chinese postman", *Math. Program.*, 5, pp. 88–124, 1973.

H.A. Eiselt, M. Gendgreau and G. Laporte, "Arc routing problems, part 1: the Chinese postman problem", *Oper. Res.*, 43, pp. 231–242, 1995a.

H.A. Eiselt, M. Gendreau and G. Laporte, "Arc routing problems, part 2: the rural postman problem", *Oper. Res.*, 43, pp. 399–414, 1995b.

G. Ghiani and G. Improta, "An algorithm for the hierarchical Chinese postman problem", *Oper. Res. Lett.*, 26, pp. 27–32, 2000.

B. Golden and R. Wong, "Capacitated arc routing problems", *Networks*, 11, pp. 305–315, 1981.

C. Malandraki and M.S. Daskin, "The maximum benefit Chinese postman problem and the maximum benefit traveling salesman problem", *Eur. J. Oper. Res.*, 65, pp. 218–234, 1993.

W.L. Pearn, "New lower bounds for the capacitated arc routing problem", *Networks*, 18, pp. 181–191, 1988.

W.L. Pearn, "Approximate solutions for the capacitated arc routing problem", *Comput. Oper. Res.*, 16, pp. 589–600, 1989.

W.L. Pearn, "Augment-Insert algorithms for the capacitated arc routing problem", *Comput. & Oper. Res.*, 18, pp. 189–198, 1991.

W.L. Pearn, "Solvable cases of the *k*-person Chinese postman problem", *Oper. Res. Lett.*, 16, pp. 241–244, 1994.

W.L. Pearn, A.A. Assad and B.L. Golden, "Transforming arc routing into node routing problem", *Comput. Oper. Res.*, 14, pp. 285–288, 1987.

W.L. Pearn and J.B. Chou, "Improved solutions for the Chinese postman problem on mixed networks", *Comput. Oper. Res.*, 26, pp. 819–827, 1999.

W.L. Pearn and M.L. Li, "Algorithms for the windy postman problem", *Comput. Oper. Res.*, 21, pp. 641–651, 1994.

W.L. Pearn and C.M. Liu, "Algorithms for the Chinese postman problem on mixed networks", *Comput. Oper. Res.*, 22, pp. 479–489, 1995.

W.L. Pearn and K.H. Wang, "On the maximum benefit Chinese postman problem", *OMEGA, Int. J. Manag. Sci.,* 31, pp. 269–273, 2003.

W.L. Pearn and T.C. Wu, "Algorithms for the rural postman problem", *Comput. Oper. Res.*, 22, pp. 819–823, 1995.

W. L. Pearn is a Professor of operations research and quality assurance at the Department of Industrial Engineering and Management, National Chiao Tung University, Taiwan, ROC. His research areas include process capability analysis, network optimization, queuing service management, applied statistics, and semiconductor manufacturing scheduling.

W. C. Chiu received her MS degree in Transportation Engineering from National Chiao Tung University. Currently, she is a PhD candidate at the Department of Industrial Engineering, National Tsing Hua University, Taiwan, ROC.