ELSEVIER

European Journal of Operational Research 142 (2002) 70–80

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Discrete Optimization

A GRASP heuristic for the mixed Chinese postman problem

A. Corberán [a,*], R. Martí [a], J.M. Sanchis [b]

[a] *DEIO, Faculty of Mathematics, University of Valencia, Dr. Moliner 50, 46100 Burjassot, Valencia Spain*
[b] *Department of Applied Mathematics, University Polytechnic of Valencia, Valencia Spain*

Received 6 May 2000; accepted 23 July 2001

Abstract

Arc routing problems (ARPs) consist of finding a traversal on a graph satisfying some conditions related to the links of the graph. In the Chinese postman problem (CPP) the aim is to find a minimum cost tour (closed walk) traversing all the links of the graph at least once. Both the Undirected CPP, where all the links are edges that can be traversed in both ways, and the Directed CPP, where all the links are arcs that must be traversed in a specified way, are known to be polynomially solvable. However, if we deal with a mixed graph (having edges and arcs), the problem turns out to be $\mathcal{NP}$-hard. In this paper, we present a heuristic algorithm for this problem, the so-called Mixed CPP (MCPP), based on greedy randomized adaptive search procedure (GRASP) techniques. The algorithm has been tested and compared with other known and recent methods from the literature on a wide collection of randomly generated instances, with up to 200 nodes and 600 links, producing encouraging computational results. As far as we know, this is the best heuristic algorithm for the MCPP, with respect to solution quality, published up to now.

Keywords: Metaheuristics; Routing; Chinese postman; GRASP

1. Introduction

The abundance of optimization problems encountered in practical settings has motivated the development of powerful optimization techniques in the last few years. Research in optimization has concentrated on analyzing complex problems and developing effective algorithms to solve them. The marriage between operations research and computer science has permitted the growth of the research area known as approximation algorithms, often called heuristic methods. Any new development which leads to better results in a particular problem, or to new approximation methods which may be applied to a wide range of problems, is of considerable value to science and industry.

Metaheuristics are a class of approximation methods that provide general frameworks that allow the creation of new heuristic methods to solve hard combinatorial optimization problems. They include genetic algorithms, scatter search, greedy randomized adaptive search procedures (GRASP), neural networks, tabu search and their

* Corresponding author. Tel.: +34-6386-4306; fax: +34-6386-4735.
E-mail address: angel.corberan@uv.es (A. Corberán).

0377-2217/02/$ - see front matter
PII: S0377-2217(01)00296-X

hybrids. They have been highly successful in finding optimal and near optimal solutions to many optimization problems. However, the design of a good metaheuristic continues to be an art nowadays, since it mainly depends on the skill and experience of the designer, both in solving techniques and in knowledge of the specific problem under consideration. This paper explores the metaheuristic approach called GRASP in the context of routing problems.

Arc routing problems (ARPs) have their origin in the celebrated Königsberg Bridge Problem solved by Euler. They basically consist of finding a route over the links of a graph which satisfies certain conditions related to the links. These problems have been deeply studied in the last 30 years due to:

1. the large number of real situations that can be modelled as an ARP: collection or delivery of goods, mail distribution, network maintenance (electrical lines or gas mains inspection), snow removal, garbage collection, etc. Papers by Eiselt et al. [9,10], Assad and Golden [1] and the recent book edited by Dror [6] summarize the state of the art and real-life applications of the ARPs. In most applications, there are some restrictions for the vehicle to traverse some streets in a specified way, and the problem needs to be modelled with a mixed graph. As a rule, one-way streets are represented by arcs (directed links) and two-way streets by edges (undirected links).
2. the progressive power and speed improvement of modern computers, which allows the handling of graphs corresponding to real applications with hundreds (or even thousands) of vertices and links.

In this paper we present a GRASP for the mixed Chinese postman problem (MCPP). In Section 2 the problem is defined and the previous work on it is summarized. The algorithm is described in detail in Section 3. Computational comparisons with other known methods proposed in the literature are given for a wide set of test instances in Section 4. Finally, some concluding comments are made in Section 5.

2. The problem

Given a graph G with non-negative costs associated with its links, the Chinese postman problem (CPP) consists of finding a minimum cost tour (closed walk) traversing, at least once, every link of G. This well known problem [16] was shown to be solvable in polynomial time both for undirected graphs – all the links are edges – and for directed graphs – all the links are arcs – by Edmonds and Johnson [7]. However, if the CPP is defined over a mixed graph – having simultaneously edges and arcs – the problem of traversing all the links in G with total minimum cost is $\mathcal{NP}$-hard [23]. A formal definition of the problem is as follows:

> Given a strongly connected mixed graph $G = (V, E, A)$ with vertex set V, edge set E, arc set A and a non-negative cost c_e for each $e \in E \cup A$, the MCPP is that of finding a minimum cost tour passing through each link $e \in E \cup A$ at least once.

If a (undirected, directed or mixed) graph G is *Eulerian*, there is a tour passing through each link of G exactly once. Obviously this tour is optimum. Furthermore, this (Eulerian) tour can be easily computed and, hence, graph G can itself be considered as the CPP solution. If a (undirected, directed or mixed) graph G is not *Eulerian*, the CPP can be formulated as the problem of finding a set of copies of links with minimum cost such that when added to G a Eulerian graph is obtained. As before, the 'augmented' graph formed by G plus the added copies of the links can be considered as the solution of the problem.

For the undirected CPP, a Eulerian graph can be obtained from G by adding edges to match odd degree vertices [3,7]). This can be done by finding a minimum cost perfect matching. For the directed case, if G is a strongly connected directed graph, a minimum cost Eulerian graph can be obtained by solving a directed flow problem as follows. For every vertex i, let s_i be the number of arcs entering i minus the number of arcs leaving i. Solve the minimum cost flow problem on G with supplies s_i for vertices i with $s_i > 0$ and demands $-s_i$ for vertices i with $s_i < 0$ [7,19].

In order to describe the way of dealing with the MCPP we need some definitions. Given a vertex i, d_i^- (in-degree) will denote the number of arcs entering i, d_i^+ (out-degree) will be the number of arcs leaving i and d_i (degree) will denote the number of links (arcs and edges) incident with i. A mixed graph $G = (V, E, A)$ is called *even* if all its vertices have even degree, it is called *symmetric* if $d_i^- = d_i^+$ for each vertex i and it is said to be *balanced* if, given any subset S of vertices, the difference between the number of arcs directed from S to $V \setminus S$ and the number of arcs directed from $V \setminus S$ to S is no greater than the number of (undirected) edges joining S and $V \setminus S$. In the process of solving the MCPP, besides duplicating some arcs and edges, we may also *orient* some edges by giving them a direction.

It is well known that a mixed graph G is Eulerian if and only if G is even and balanced [13]. Notice that if G is even and symmetric then G is also balanced (and Eulerian). Moreover, if G is even, the MCPP can be exactly solved in polynomial time [7]. The procedure, as described in [20], works as follows:

Even MCPP Algorithm:

1. Given an even and strongly connected mixed graph, $G = (V, E, A)$, let A_1 be the set of arcs obtained by arbitrarily assigning a direction to the edges in E and with the same costs. Compute $s_i = d_i^- - d_i^+$ for each vertex in the directed graph $(V, A \cup A_1)$. A vertex i with $s_i > 0$ ($s_i < 0$) will be considered as a source (sink) with supply (demand) s_i ($-s_i$). Note that as G is an even graph, all supplies and demands are even numbers (zero is considered an even number).
2. Let A_2 be the set of arcs with opposite direction to those in A_1 and with the costs of the corresponding edges, and let A_3 be a set of arcs parallel to that of A_2 with zero costs.
3. Solve a minimum cost flow problem to satisfy the demands of all the vertices in the graph $(V, A \cup A_1 \cup A_2 \cup A_3)$, in which each arc in $A \cup A_1 \cup A_2$ has infinite capacity and each arc in A_3 has capacity 2. Let x_{ij} be the optimal flow.
4. For each arc (i, j) in A_3 do: if $x_{ij} = 2$ then orient the corresponding edge in G from i to j (the direction, from j to i, assigned to the associated edge in step 1 was "wrong"); if $x_{ij} = 0$ then orient the corresponding edge in G from j to i (in this case, the orientation given to the corresponding edge in step 1 was the "good" one). (Note that the case $x_{ij} = 1$ is impossible as all flow values through arcs in A_3 are even numbers.)
5. Augment G by adding x_{ij} copies of each arc in $A \cup A_1 \cup A_2$. The resulting graph is even and symmetric.

Although it does not provide the optimum MCPP tour when the mixed graph $G = (V, E, A)$ is not even, the above algorithm is the basis of two heuristics suggested by Edmonds and Johnson [7] and developed and improved by Frederickson [14]. Algorithm MIXED1 would be equivalent to first transforming G into an even graph and then applying the previous *Even MCPP Algorithm*. Specifically:

Algorithm MIXED1:

(1 Evendegree) Let $G = (V, E, A)$ be a strongly connected mixed graph. Ignoring arc directions solve a minimum cost matching of odd degree vertices and augment G by adding all links used for the matching solution. The resulting graph $G' = (V, E', A')$ is an even graph.

(2 Inoutdegree) With supplies and demands computed in graph (V, A'), solve a minimum cost flow problem in G' to obtain a symmetric graph $G'' = (V, E'', A'')$. (Frederickson pointed out that after this step some vertices in G'' may have odd degree.)

(3 Evenparity) Let V_o be the set of odd degree nodes in G''. Identify cycles consisting of alternating paths in $A'' \setminus A$ and E'', with each path starting and ending at vertices in V_o. The paths in $A'' \setminus A$ will be formed without considering the arc directions. As the cycles are covered, its arcs are either duplicated or deleted and its edges are directed, so the resulting graph becomes even, while maintaining the symmetry for each vertex.

Steps 1 and 2 above were first proposed by Edmonds and Johnson [7], who showed that there is a minimum cost solution to the flow problem in Step 2 that preserves the property that each vertex has even degree. But, as Frederickson [14] pointed

out, their argument merely proves the existence of such a solution. Step 3 above is a simple linear time algorithm proposed by Frederickson to perform this task.

Algorithm MIXED2 can be considered as the reverse version of MIXED1. It first solves a minimum cost flow problem in G, with supplies and demands computed in graph (V,A), to obtain a symmetric graph (V,E',A'). In a second step, it solves the (undirected) CPP on each connected component of subgraph (V,E') to finally obtain an even and symmetric graph.

Frederickson [14] showed that both algorithms, MIXED1 and MIXED2, have a worst case ratio of 2, but the *Mixed Algorithm*, which consists of applying both algorithms consecutively and select the best tour obtained, has a worst case ratio of 5/3. See also [17] for some illustrations and other details on the above heuristic procedures.

Recently, Raghavachari and Veerasamy [24] have proposed a modification to the Frederickson's Mixed Algorithm with a better worst case ratio of 3/2:

Modified Mixed Algorithm:

- Given $G=(V,E,A)$, solve a minimum cost flow problem in G with supplies and demands computed in graph (V,A). The resulting graph is (V,E',A'), where $E'\subseteq E$ are the edges of G that were not oriented (used) by the flow solution and $A'\supseteq A$ are arcs that satisfy in-degree equal to out-degree at each vertex.
- Before running Evendegree of MIXED1 algorithm, reset the costs of all arcs and edges in A' to 0, forcing Evendegree of MIXED1 to duplicate edges and arcs in A' whenever possible.
- Use the original costs for the rest of MIXED1 algorithm.
- There are no changes in the MIXED2 algorithm.

With respect to exact algorithms, Christofides et al. [4] proposed a formulation with a variable for each arc, two variables for each edge (representing the number of times it is traversed in either direction) and a variable for each vertex. Then a Branch and Bound algorithm was implemented and applied to a set of 34 randomly generated instances with $7\leqslant|V|\leqslant 50$, $3\leqslant|A|\leqslant 85$ and $4\leqslant|E|\leqslant 39$. All these instances were solved to optimality.

Grötschel and Win [15] have proposed a cutting-plane procedure to solve the Windy postman problem (WPP) that can also be applied to the MCPP. With this algorithm, Grötschel and Win [15] solved to optimality the nine MCPP instances tried, with $52\leqslant|V|\leqslant 172$, $31\leqslant|A|\leqslant 116$ and $37\leqslant|E|\leqslant 154$.

Nobert and Picard [21] present a cutting-plane algorithm that exactly solves the MCPP based in the necessary and sufficient condition for a mixed graph to be Eulerian given by Ford and Fulkerson [13]. In their formulation, and for the first time, only one variable is associated with each edge of G. They also proved an important result: balanced-set inequalities, modelling the balanced sets condition, can be separated in polynomial time (see also [2] for a direct and short proof of this nice result). With this algorithm Nobert and Picard [21] solved 148 instances out of 180 randomly generated instances, with $10\leqslant|V|\leqslant 169$, $2\leqslant|A|\leqslant 2876$ and $15\leqslant|E|\leqslant 1849$.

Corberán et al. [5] have implemented a preliminary cutting plane algorithm for the general routing problem (GRP) defined on a mixed graph and have studied its associated polyhedron. Mixed GRP includes the MCPP as a particular case and some of the polyhedral results and the GRP algorithm can also be applied to the MCPP. Lower bounds presented in Section 4 were obtained with this cutting-plane algorithm.

Finally, polyhedral results directly related to the MCPP can be found in [8].

3. Proposed procedure

GRASP is by now a well known metaheuristic for solving hard combinatorial optimization problems. The origins of this technique can be traced back to [11]. Numerous applications of this method have appeared (see, for instance, [18,25]), along with tutorial papers designed to expand the knowledge related to this methodology [12].

Each GRASP iteration consists of two phases, a construction phase and a local search phase. The

construction phase is iterative, greedy and adaptive. A feasible solution is iteratively constructed, one element at a time according to a greedy function evaluation. The method is adaptive in the sense of updating relevant information from one construction step to the next. Therefore, the greedy evaluation, associated with each element at each iteration, is modified to reflect the changes brought on by the selection of the previous element.

Since the solutions generated by a GRASP construction are not guaranteed to be locally optimal, a local search is usually applied to improve each constructed solution. GRASP implementations are generally robust in the sense that it is difficult to find or devise pathological instances for which the method will perform arbitrarily badly. It has been found that the sampling distribution of the solution space performed by GRASP has a mean value that can be worse than the one obtained with a deterministic construction, but the best over all trials dominates this solution with a high probability. We now describe the details of our GRASP implementation for the MCPP.

3.1. Construction phase

Given a mixed graph, an edge is oriented if we assign a direction to it. Then it can be considered an arc since it must be traversed in the assigned direction. If all the edges in the mixed graph are oriented, then the graph can be considered as a directed graph. As has been described in the previous section, the CPP on a directed graph is polynomialy solvable by a minimum cost flow problem.

Basically, our construction method orients the edges of the graph in order to obtain a directed graph in which a CPP is solved. In this way we obtain a solution to the MCPP in the original mixed graph. The construction phase starts by creating a list of non-oriented edges (U) which at the beginning consists of all the edges in the graph (initially $U = E$). At each construction step, an edge is selected and oriented. The set of oriented edges will be denoted by E_d.

From now on, let $d(v)$ be the difference between the number of arcs and oriented edges with end point in v and the number of arcs and oriented edges with initial point in v. For the sake of simplicity, $d(v)$ will be called the degree of vertex v. The greedy evaluation $w(i,j)$ of the edge (i,j) represents, in a certain sense, the convenience of orienting it. Consider, for instance, an edge (i,j) such that $d(i) = 3$ and $d(j) = -4$. Then, it seems more appropriate to orient this edge from i to j than in the other direction (obviously this is a "local" argument that depends on the available information at a given step). On the other hand, if $d(i) = 2$ and $d(j) = 2$ there is no information indicating a direction for the edge, and then it would be better to wait until a later iteration in which other incident edges with i and j have been oriented. We propose the following evaluation $w(i,j)$ of the edge (i,j), where ϵ represents a very low positive quantity:

Case 1: If $d(i)d(j) > 0$ then $w(i,j) = -|d(i) + d(j)|$.
Case 2: If $d(i)d(j) < 0$ then $w(i,j) = |d(i) - d(j)|$.
Case 3: If $d(i) = 0$ and $d(j) = 0$ then $w(i,j) = -\epsilon$.
Case 4: If $d(i) = 0$ or $d(j) = 0$ then $w(i,j) = \epsilon$.

This edge evaluation ranks the edges according to the amount of current information relative to orient each one of them. Then, the set U is ordered according to w, with the first edge the one with maximum w-value. At each construction step, an edge (i,j) is randomly selected from the first s elements in U. Then edge (i,j) is oriented from the vertex with higher degree to the vertex with lower degree; if i and j have the same degree, edge (i,j) is randomly oriented. Sets U and E_d are updated by deleting and adding (i,j), respectively, and $d(i)$, $d(j)$ and $w(u,v)$ are also updated for all edges $(u,v) \in U$ incident with i or j. The construction phase terminates after $|E|$ steps, when all the edges have been selected and oriented. Then, a minimum cost flow problem with demands and supplies computed with respect to the arcs in $A \cup E_d$ is solved on the directed graph $(V, A \cup E_d \cup E'_d)$, where E'_d is the set of arcs parallel to those in E_d with the same costs but with opposite directions. Adding the arcs in the flow solution to the graph

$(V, A \cup E_d)$, an Eulerian digraph is obtained. This graph corresponds to a solution (tour) in the original mixed graph.

It should be noted that there are some cases in which there is only one possible orientation for an edge. Specifically, consider a vertex i incident with only one edge $(i,j) \in U$ and with no arc or oriented edge leaving (entering) it. Then (i,j) must be directed from i to j (respectively from j to i), since otherwise the cost of the final solution should be greater. Therefore, at each construction step, once an edge has been oriented, we check this condition on its both end vertices in order to orient, if it is possible, another incident edge.

3.2. Improvement phase

In this phase a local search procedure is applied with the objective of finding a locally optimal solution that may be better than the constructed solution. The procedure starts from the MCPP tour obtained in the constructive phase. It is represented by a strongly connected digraph, containing all the original arcs and edges (oriented) of the graph and some duplications of these links (that can appear more than once).

A simple but effective step to improve the solution obtained in the constructive phase follows. Delete any two copies of any edge appearing more than twice in the solution with opposite orientations. Note that in the solution graph there is either a single arc, two arcs with opposite directions or two or more arcs, all of them with the same direction associated to each original edge $e \in E$.

Switches are used as the primary mechanism in the local search algorithm to move from one solution to another. Given two vertices i, j linked by a path of duplications, we define move $m(i,j)$ as deleting this path of duplications from the current solution and adding the shortest path in G joining both vertices (i.e. a duplication of each link in the path). The move value is the difference between the objective function values before and after the move. It can be computed as the cost of the deleted path minus the cost of the added shortest path.

Each step of the improvement phase consists of selecting a pair of vertices to be considered for a move. We have restricted our attention to some pairs of vertices in order to reduce the computational effort associated with each move. Obviously, only vertices joined by a path of duplications are suitable for moves. The following procedure selects those "promising" pairs:

From the solution graph G_s, a new mixed graph $G' = (V, E', A')$ is constructed as follows. For each original arc $(i,j) \in A$ we put $n-1$ copies of it in A', where n is the number of times it appears in G_s. For each original edge $e \in E$ having exactly two arcs with opposite directions associated in G_s we put an edge in E'; otherwise, $n-1$ copies of its associated arc are added to A', where again n is the number of times the arc appears in G_s.

Then, we compute the (non-trivial) connected components of graph G' and reduce the search to pairs of vertices inside each component. These components are acyclic graphs (with eventually several arcs joining two adjacent vertices) since they come from the solution of a minimum cost flow problem. In order to reduce the computational effort, we restrict our attention to those vertices adjacent to only one other vertex (like the leaves in a tree graph). We have experimentally found that, in most cases, long paths join these vertices, so it is expected that good moves be associated to these pairs of vertices. If the move has a positive value, it is performed; otherwise it is rejected. An improvement phase terminates when all pairs of these vertices have been considered within each component and no further reduction can be performed.

It is worth mentioning that when a move is performed, the connected components can be modified since we remove one path and add another one to the graph. We have implemented a multiple pointer structure to efficiently handle those changes in a dynamic way.

After completing *Niter* consecutive GRASP iterations without any improvement, the best solution found so far is returned to as the final solution. The following outline summarizes the algorithm:

Do while (iter < Niter)

Construction phase:

(1) iter = iter + 1. Let $E_d = \emptyset$ and $U = E$. For each edge $e = (i,j) \in U$ compute weight$(e) = w(i,j)$.

(2) Repeat while $U \neq \emptyset$:

(2.1) Consider the s edges in U with greater weight, where s is a given parameter (candidate list (CL) length).

(2.2) Select one of them randomly, say (i,j), and delete it from U. Edge (i,j) is oriented from the vertex with higher degree to the vertex with lower degree; it will be randomly oriented if i and j have the same degree. Label it as 'oriented from i to j' and add (i,j) to E_d, or as 'oriented from j to i' and add (j,i) to E_d.

(2.3) Update the *weights* of all edges in U incident with i and j.

(3) Compute the demand/supply of each vertex with respect to the arcs and oriented edges.

(4) Solve a minimum cost flow problem on a directed graph having: one arc (i,j) for each $(i,j) \in A$ with the same cost, two arcs (i,j), (j,i) for each edge $(i,j) \in E$ with the same cost and the demands/supplies computed in (3).

(5) Add to $A \cup E_d$ one copy of each arc and each oriented edge used by the flow (4) to obtain a Eulerian directed graph. Associated to this graph there is a tour T for the MCPP on G with cost $z(T)$.

Improvement phase: Do while (Improvement > 0)

(6) Remove from T any two copies of every edge e appearing more than twice in T and having opposite 'orient-labels'. Update $z(T)$.

(7) Construct the mixed graph $G' = (V, E', A')$ as has previously been defined and compute its connected components.

(8) For each one of these connected components: find a pair of nodes u and v such that the length of the shortest path from u to v in G, say $\mathscr{P}$, is less than the length of the shortest path from u to v, $\mathscr{Q}$, in this connected component. Remove from T the (copies of) links in $\mathscr{Q}$ and add to T (one copy of) each link in $\mathscr{P}$. Set Improvement = length$(\mathscr{Q})$ − length$(\mathscr{P})$ and $z(T) := z(T) -$ Improvement.

Termination Test:

(9) If $(z(T) < z_{\text{best}})$ then $z_{\text{best}} = z(T)$, store T as T_{best} and do iter = 0.

Write T_{best} and z_{best}.

4. Computational experiments

The procedure described in the previous section was implemented in C, and all the experiments were performed on a Pentium III 700 MHz personal computer. Before testing the effectiveness of our procedure, we performed some preliminary experiments to explore the effect of changes in the search parameter CL length. We have also tested the algorithm with some variants in the edge function evaluation w given in the construction phase.

The graph generator works as follows. Given a number n of vertices, the number of links is randomly chosen between $2n$ and $3n$. This total number of links is multiplied by the desired percentage of edges in order to determine the number of edges. Costs of links are randomly selected between 1 and a random number in the interval (20, 30). Then we check if the graph is strongly connected or not. If it is not, additional links are added until obtaining a strongly connected graph. We use this generator to create 225 instances with the following characteristics. For each combination of 50, 100 and 200 vertices and 30%, 50% and 70% of arcs, 25 instances were generated. By 30% of arcs, for example, we mean that the 30% of all generated links in a given instance are arcs.

In order to measure the quality of the proposed method, we have computed its deviation (*Dev.*) from a lower bound. This lower bound has been obtained with a preliminary cutting-plane algorithm devised for the GRP on a mixed graph [5]. This cutting-plane algorithm includes separation procedures for the odd cut and balanced-set inequalities. Therefore, in what MCPP respects, it can be considered equivalent to the algorithm in [21]. The initial LP relaxation includes one odd cut inequality for each odd vertex and one balanced-set inequality for each 'unbalanced' vertex. The separation problem associated with odd cut inequalities is solvable in polynomial time by means

Table 1
Experiments with several CL lengths

CL length	>0	1	5	10	20	1000
% Dev.	1.53	2.83	1.31	1.15	1.30	1.58
Time (seconds)	12.39	0.13	11.57	15.51	10.95	12.04

of the Padberg and Rao [22] procedure to find minimum odd cut-sets. Standard heuristics were also implemented, as well as the exact method for identifying violated balanced-set inequalities described in [21].

We have considered five different values for the length of the restricted CL in the construction phase (parameter *CL length*). A value of 1 means that the selection is made on a list of only one element, which is equivalent to a pure greedy algorithm. On the other hand, a value of 1000 means that the random selection is made over the whole list of non-oriented edges, which is equivalent to a pure random method. Between both extreme values we have considered three regular values for the list length: 5, 10 and 20. We have also tested another variant (labelled as >0) given by restricting the list to those elements with a positive value of $w(i,j)$. These 6 variants have been compared on a set of 18 instances of different sizes and arc percentages. The results of this experiment are reported in Table 1, where the number of total iterations *Niter* has been set to 100.

Table 1 shows that the best results are obtained with CL length = 10, although this method requires more computational effort. As was expected, the worst results correspond to the greedy and pure random strategies. Since the greedy option (CL length = 1) produces the same solution in all the runs, we have only performed one run for this method (Niter = 1). Therefore, it presents a significant lower run time than the other procedures.

Now we compare some variants in the edge evaluation w given in the construction phase of the algorithm. Let $w(i,j)$ be the evaluation function defined in the previous section, and let $w'(i,j)$ be the new evaluation function that replaces the old one in the algorithm. We have considered four different variants that try to emphasize or to limit the effect of edge weights according to their costs:

[Variant 2:] $w'(i,j) = w(i,j)c(i,j)$

[Variant 3:] $w'(i,j) = w(i,j)/c(i,j)$

[Variant 4:] $w'(i,j) = w(i,j)c(i,j)$ if $w(i,j) \geqslant 0$; $w'(i,j) = w(i,j)/c(i,j)$ otherwise

[Variant 5:] $w'(i,j) = w(i,j)c(i,j)$ if $w(i,j) < 0$; $w'(i,j) = w(i,j)/c(i,j)$ otherwise.

The $w(i,j)$ value reflects the convenience of edge (i,j) to be oriented. We believe that edges with a great cost will have a great influence in the solution cost. In Variant 2 these edges are selected first to be oriented, while in Variant 3 their orientation is left to final steps of the construction process. Variants 4 and 5 propose mixed strategies of Variants 2 and 3 according to the sign of the $w(i,j)$.

Table 2 reports the results of these 4 variants for different values of the parameter *CL length* over the same set of instances. For each variant, a similar behavior of *Dev.* and *Time* for the different values of parameter *CL length* is shown. The best results are obtained for *CL length* within the range [5,19], while the worst ones are obtained with

Table 2
Experiments with different $w(i,j)$ functions

	CL length	>0	1	5	10	20	1000
Variant 2	% Dev.	1.48	2.84	1.36	1.37	1.41	1.60
	Time (seconds)	12.63	0.16	12.40	12.23	10.97	14.90
Variant 3	% Dev.	1.46	3.31	1.17	1.32	1.26	1.83
	Time (seconds)	16.39	0.14	13.37	16.95	16.36	10.31
Variant 4	% Dev.	1.49	3.04	1.43	1.26	1.33	1.53
	Time (seconds)	13.88	0.12	12.97	10.49	9.19	10.87
Variant 5	% Dev.	1.62	3.41	1.31	1.32	1.24	1.72
	Time (seconds)	10.47	0.13	14.04	11.82	12.34	11.75

CL length = 1000 (except, obviously, with the greedy option).

Tables 1 and 2 show that the best solution quality is obtained by the algorithm with the original evaluation w and a value for parameter *CL length* equal to 10, considering its average percent deviation from the lower bound of 1.15. Then, we will use this variant in the following computational experiments, where the value *Niter* has been set to 1000.

In order to compare our algorithm to other heuristics, we have implemented the classical Mixed 1 and Mixed 2 algorithms [14] as well as the very recent modification of Mixed 1 proposed by Raghavachari and Veerasamy [24] (denoted by M. Mixed 1).

Table 3 reports the percentage deviation from the lower bound and the running times of the four methods considered in each set of instances. Each set contains 75 instances of 50, 100 and 200 vertices, respectively, divided into 3 groups of 25 instances according to the percentage of arcs. The average number of links for each set is shown in the column *Links*.

Table 3 shows that the GRASP procedure performs quite well on graphs with 50 vertices since it presents an average percentage deviation from the lower bound of 0.56% achieved in an average of 3.4 seconds. For these instances, the performance of this heuristic, although slightly better when the percentage of arcs is 30%, may be considered as similar for all instance characteristics. The algorithm also presents good results on larger instances although, as was expected, percentage deviation and running time increase with the number of vertices and links (1.13% on 40.9 seconds for graphs with 100 vertices and 1.67% on 356.5 seconds for graphs with 200 vertices). In addition, Table 3 shows the algorithm's expected behavior with respect to the percentage of arcs. In these larger instances better deviation percentages and lower running times are associated with larger percentages of arcs. It can easily be explained by the fact that our procedure is based primarily on the assignment of a direction to every edge in the graph until a fully directed graph is obtained.

These experiments also reveal that M. Mixed 1 is significantly better than the original Mixed 1 algorithm, except for those instances with a lower percentage of arcs. Moreover, its performance slightly improves when the percentage of arcs increases, while in the original Mixed 1 the quality of the solutions clearly deteriorates as this percentage increases. On the other hand, Mixed 2 presents the worst results, except for those instances with a large number of arcs.

Table 4 summarizes the results obtained with the *Mixed Algorithm*, the *Modified Mixed Algorithm* and the GRASP heuristic. As it was mentioned in Section 2, the *Mixed Algorithm* consists

Table 3
Computational experiments with different instance characteristics

V	Links	Arcs (%)	Mixed 1		Mixed 2		M. Mixed 1		GRASP	
			% Dev.	Time	% Dev.	Time	% Dev.	Time	% Dev.	Time
		30	1.93	0.0	16.63	0.0	3.18	0.0	0.42	6.2
50	132	50	7.10	0.0	20.41	0.0	3.15	0.0	0.60	2.8
		70	11.25	0.0	11.18	0.0	1.84	0.0	0.66	1.2
Avg.			6.76	0.0	16.07	0.0	2.72	0.0	0.56	3.4
		30	2.71	0.1	17.99	0.0	2.75	0.1	1.34	89.4
100	257	50	6.93	0.1	19.83	0.0	2.51	0.1	1.36	25.9
		70	11.10	0.1	10.16	0.0	1.83	0.1	0.69	7.4
Avg.			6.91	0.1	15.99	0.0	2.36	0.1	1.13	40.9
		30	3.14	0.3	18.92	0.2	3.39	0.2	2.04	812.5
200	510	50	7.62	0.3	19.67	0.2	2.62	0.2	2.06	209.6
		70	11.15	0.3	9.43	0.1	2.07	0.2	0.90	47.3
Avg.			7.30	0.3	16.01	0.2	2.69	0.2	1.67	356.5

Table 4
Comparisons with the best solution found

	Mixed Algorithm (Frederickson)	Modified Mixed Algorithm (R and V)	GRASP heuristic
Average objective value	5267.14	5077.18	5016.70
Number of best	20	28	184
Deviation from best (%)	5.04	1.58	0.11

of running both Mixed 1 and Mixed 2 and select the best solution obtained. Similarly, the *Modified Mixed Algorithm* selects the best solution between those provided by M. Mixed 1 and Mixed 2. This table reports the average objective value, the number of best solutions found and the average percentage deviation from the best solution found with the three methods.

Table 4 shows that the best solution quality is obtained by the GRASP method, which is able to match 184 best solutions out of 225 solutions, while the *Mixed Algorithm* and the *Modified Mixed Algorithm* match 20 and 28 best solutions, respectively. However, GRASP employs more computational time than the other methods. Besides its good theoretical behavior, the *Modified Mixed Algorithm* is very competitive in practice, considering its average percent deviation from the lower bound and from the best solution known and its short running times.

Average deviations from the lower bound are illustrated in three separate Figs. 1–3, one for each set of instances. These figures clearly show the behavior of the three algorithms with respect to the instances size and the percentage of arcs.

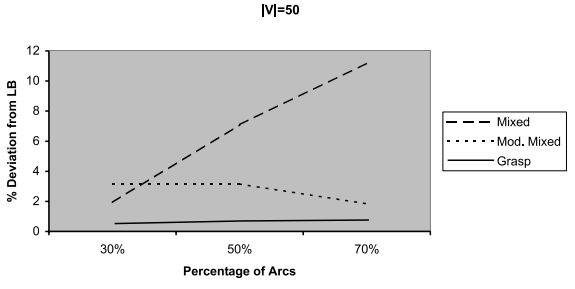

Fig. 1.

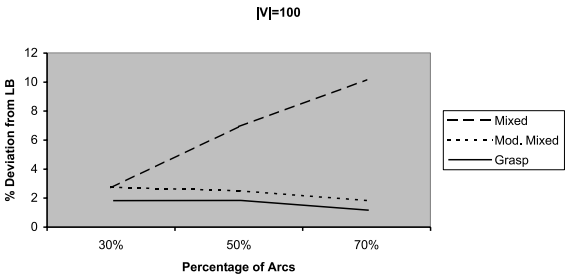

Fig. 2.

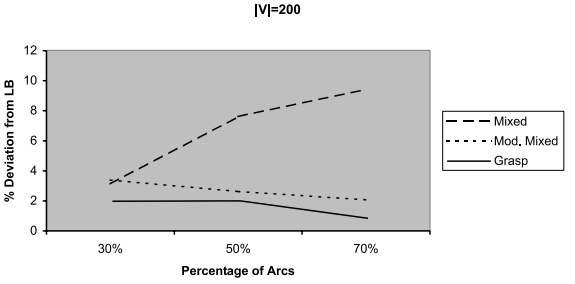

Fig. 3.

5. Conclusions

We have developed a heuristic procedure for the MCPP. The procedure is based on the GRASP methodology. Computational comparisons with a lower bound and with other known heuristics have been performed in order to study the efficiency of the algorithm.

Computational testing was performed on a set of 225 randomly generated graphs with up to 200 vertices and 600 links. Comparisons were made with a lower bound that has been shown to be relatively close to the optimal solution. This lower bound was obtained using a cutting-plane algorithm that is equivalent to the exact procedure proposed by Nobert and Picard [21]. Other comparisons have been made with two very known heuristics from the literature, the classical *Mixed Algorithm* [14] and the *Modified Mixed Algorithm* [24]. The computational experiments show that our GRASP procedure outperforms these other approaches and provides results close to the lower bound value (with an average deviation of 1.12%) in all these instances.

References

[1] A.A. Assad, B.L. Golden, Arc routing methods and applications, in: M.O. Ball, T.L. Magnanti, C.L. Monma, G.L. Nemhauser (Eds.), Network Routing. Handbooks of Operations Research and Management Science, vol. 8, North-Holland, Amsterdam, 1995, pp. 375–483.

[2] E. Benavent, A. Corberán, J.M. Sanchis, Linear programming based methods for solving arc routing problems, in: M. Dror (Ed.), ARC Routing: Theory, Solutions and Applications, Kluwer Academic Publishers, Reading, MA, 2000, pp. 231–276.

[3] N. Christofides, The optimum traversal of a graph, Omega 1 (6) (1973) 719–732.

[4] N. Christofides, E. Benavent, V. Campos, A. Corberán, E. Mota, An optimal method for the mixed postman problem, in: P. Thoft-Christensen (Ed.), System Modelling and Optimization, Lecture Notes in Control and Information Sciences, vol. 59, Springer, Berlin, 1984, pp. 641–649.

[5] A. Corberán, A. Romero, J.M. Sanchis, The general routing problem on a mixed graph, Technical Paper, Department of Statistics and OR, University of Valencia, Spain, 1999.

[6] M. Dror (Ed.), ARC Routing: Theory, Solutions and Applications, Kluwer Academic Publishers, Reading, MA, 2000.

[7] J. Edmonds, E.L. Johnson, Matching Euler tours and the Chinese postman, Mathematical Programming 5 (1973) 88–124.

[8] R.W. Eglese, A.N. Letchford, Polyhedral theory for arc routing problems, in: M. Dror (Ed.), ARC Routing: Theory, Solutions and Applications, Kluwer Academic Publishers, Reading, MA, 2000, pp. 199–230.

[9] H.A. Eiselt, M. Gendreau, G. Laporte, Arc routing problems, Part 1: The Chinese postman problem, Operations Research 43 (1995) 231–242.

[10] H.A. Eiselt, M. Gendreau, G. Laporte, Arc routing problems, Part 2: The rural postman problem, Operations Research 43 (1995) 399–414.

[11] T. Feo, M. Resende, A probabilistic heuristic for a computationally difficult set covering problem, Operations Research Letters 8 (1989) 67–71.

[12] T. Feo, M. Resende, Greedy randomized adaptive search procedures, Journal of Global Optimization 6 (1995) 109–133.

[13] L.R. Ford, D.R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, NJ, 1962.

[14] G.N. Frederickson, Approximation algorithms for some postman problems, Journal of the ACM 26 (3) (1979) 538–554.

[15] M. Grötschel, Z. Win, A cutting-plane algorithm for the windy postman problem, Mathematical Programming 55 (1992) 339–358.

[16] M. Guan, Graphic programming using odd or even points, Chinese Mathematics 1 (1962) 237–277.

[17] A. Hertz, M. Mittaz, Heuristic algorithms, in: M. Dror (Ed.), ARC Routing: Theory, Solutions and Applications, Kluwer Academic Publishers, Reading, MA, 2000, pp. 327–387.

[18] M. Laguna, T. Feo, H. Elrod, A greedy randomized adaptive search procedure for the 2-partition problem, Operations Research 42 (4) (1994) 677–687.

[19] T.M. Liebling, Graphentheorie in planungs – und tourenproblemen, in: Lecture Notes in Operations Research and Mathematical Systems, vol. 21, Springer, Berlin, 1970.

[20] E. Minieka, Optimization Algorithms for Networks and Graphs, Marcel Dekker, New York, 1979.

[21] Y. Nobert, J.C. Picard, An optimal algorithm for the mixed Chinese postman problem, Networks 27 (1996) 95–108.

[22] M.W. Padberg, M.R. Rao, Odd minimum cut-sets and *b*-matchings, Mathematics for Operations Research 7 (1982) 67–80.

[23] C.H. Papadimitriou, On the complexity of edge traversing, Journal of the ACM 23 (1976) 544–554.

[24] B. Raghavachari, J. Veerasamy, Approximation algorithms for the mixed postman problem, in: R.E. Bixby, E.A. Boyd, R.Z. Ríos-Mercado (Eds.), Proceedings of 6th Integer Programming and Combinatorial Optimization, Springer, Berlin, 1998, pp. 169–179.

[25] M. Resende, Computing approximate solutions of the maximum covering problem with GRASP, Journal of Heuristics 4 (1998) 161–177.