Journal of Heuristics, 11: 233–257, 2005

Grasp Embedded Scatter Search for the Multicommodity Capacitated Network Design Problem

ADA M. ALVAREZ
Facultad de Ingeniería Mecánica y Eléctrica, Universidad Autónoma de Nuevo León, Monterrey, NL, México
email: adita@yalma.fime.uanl.mx

JOSÉ LUIS GONZÁLEZ-VELARDE
Instituto Tecnológico y de Estudios Superiores de Monterrey, Monterrey, NL, México
email: gonzalez.velarde@itesm.mx

KARIM DE-ALBA
Facultad de Ingeniería Mecánica y Eléctrica, Universidad Autónoma de Nuevo León, Monterrey, NL, México
email: karim@yalma.fime.uanl.mx

Submitted in March 2004 and accepted by Steve Chiu in April 2005 after 1 revision

Abstract

A GRASP embedded Scatter Search is developed for the multicommodity capacitated network design problem. Difficulty for this problem arises from the fact that selection of the optimal network design is an NP-complete combinatorial problem. There exist no polynomial exact algorithms which can solve this problem in a reasonable period of time for realistically sized instances. In such cases, heuristic procedures are commonly used. Two strategies were designed for GRASP: a traditional approach and a memory based technique. As for Scatter Search, 5 different strategies were used to update the reference set. Computational results on a large set of randomly generated instances show the convenience of the proposed procedures.

Key Words: network design, Scater Search, GRASP

1. Introduction

The network design problem addresses the following basic question: what configuration of the network minimizes the sum of the fixed costs of edges chosen to be in the network and the costs of routing goods through the network defined by these edges?

These problems arise in applications as diverse as capital investment decision making for transportation planning, vehicle routing and vehicle fleet planning, material handling system design, design of telecommunication networks, facility location and design of freight distribution system. For a comprehensive survey of network design problems and their applications, see Magnanti and Wong (1984).

Because of its usefulness in these applications, much research has been carried out in the area of network design. This problem is NP complete (Johnson, Lenstra, and Rinooy, 1978; Hochbaum and Segev, 1989) and most of the solution methods are based on the above observation. In particular, the uncapacitated version of this type of problem has been quite well studied in several research reports (Magnanti, Mireault, and Wong, 1986; Balakrishnan and Magnanti, 1989; Holmberg and Hellstrand, 1998). However, compared to the uncapacitated case, few references can be found in the literature for capacitated network design. While in some cases the network is, or can be assumed to be, uncapacitated, a capacitated model is more general and often much more suitable because capacity constraints often do arise in real-life applications.

In this paper, the fixed charge, capacitated, multicommodity network design problem is addressed. Here, several commodities (goods, data packets, people, etc.) have to be moved over the links of the networks which have limited capacities from their respective origins to their respective destinations. Furthermore, in addition to the transportation costs related to the volume of each commodity flowing through a given link, a fixed (construction or utilization) cost is paid only once as soon as a link is used.

The tradeoff between the variable and fixed costs inherent in the selection of any solution, as well as the interplay between the limited capacity (and the resulting competition among the various commodities) and the fixed costs of using the links of the network, cause serious obstacles when solving realistically sized instances.

Several techniques have been proposed to solve the capacitated network design problems including Lagrangian relaxation (Gendron and Crainic, 1994b; Holmberg and Yuan, 2000), bounding procedures (Gendron and Crainic, 1994a, 1996; Crainic, Frangioni, and Gendron, 2001), Tabu Search (Crainic, Gendreau, and Farvolden, 2000), and enumerative algorithms (Sridhar and Park, 2000). Nevertheless, these results can not be applied to the problem addressed here due to the fact that these works consider directed networks, associating a capacity to *each* edge direction. In the present case, the edge capacity is shared between every commodity flowing through that edge without regarding of the direction (as it is the case in telecommunication networks), so it is impossible to transform our network into a directed one to use the methods designed especially for this kind of networks.

The only work found by the authors which tackle the problem presented in this paper is the one by Herrmann et al. (1996) who considered the fixed charge capacitated network design problem on undirected graphs and presented a dual ascent approach for finding lower bounds and near-optimal solutions. Even when this work was applied to small grid networks, its results were not satisfying enough.

A comparison between Dual Ascent method and the method designed here was carried out and the results are presented in Table 5 (see Section 5.2.3).

Taking into account what it was said, the contribution of this paper is to present an efficient procedure to find good feasible solutions to realistically sized capacitated multicommodity fixed cost network design problems. The procedure may be classified as a hybrid metaheuristic and is based on the Scatter Search and GRASP metaheuristics which have been successfully used in several applications (Pacheco and Casado, 2004; Delgado, Laguna, and Pacheco, 2004).

2. Problem formulation

Given the following parameters

N	Set of nodes
A	Set of undirected arcs available for designing the network
E	Set of available edges
(i,j)	Directed arc from node "i" to node "j"
$\{i,j\}$	Edge (undirected arc) between nodes "i" and "j"
K	Set of commodities
$O(k)$	Origin of commodity k
$D(k)$	Destination of commodity k
F_{ij}	Fixed cost for using edge $\{i,j\}$ in the network design
c_{ij}^k	Variable cost for transporting one unit of flow of commodity k through arc (i,j).
d^k	Demand for commodity k
u_{ij}	Edge $\{i,j\}$ capacity

and the following variables

y_{ij}	Decision variable indicating if edge $\{i,j\}$ is included in the final network design
x_{ij}^k	Flow on arc (i,j) for commodity k

The multicommodity capacitated network design problem (MCND) can be formulated as follows:

$$\min \left[\left(\sum_{k \in K} \sum_{(i,j) \in A} c_{ij}^k x_{ij}^k \right) + \sum_{\{i,j\} \in E} F_{ij} y_{ij} \right] \tag{1}$$

$$\sum_{\{j:(i,j) \in A\}} x_{ij}^k - \sum_{\{j:(j,i) \in A\}} x_{ji}^k = \begin{cases} d^k & \text{if } i = O(k) \\ -d^k & \text{if } i = D(k) \\ 0 & \text{otherwise} \end{cases} \quad \forall\, k \in K\,,\ \forall\, i \in N \tag{2}$$

$$\sum_{k \in K} \left(x_{ij}^k + x_{ji}^k \right) \leq u_{ij} y_{ij} \qquad \forall\, \{i,\, j\} \in E \tag{3}$$

$$x_{ij}^k \geq 0 \qquad \forall\, k \in K; \forall (i,\, j) \in A \tag{4}$$

$$y_{ij} \in \{0,\, 1\} \qquad \forall\, \{i,j\} \in E$$

This objective (1) captures the basic tradeoff between routing cost savings and fixed costs for using network edges. A larger network may cost more to build, but may reduce operating costs by including more attractive origin-destination paths. Conversely, a smaller network may increase the operating costs. The standard flow conservation equations (one for each commodity k and each node i) are represented in (2). In (3), flow for all the commodities that circulate in any direction of the edge $\{i,\, j\}$ can not exceed the capacity of this edge. This set of constraints prohibits flow through inactive edges, i.e. edges $\{i,\, j\}$ not included in the design ($y_{ij} = 0$) And finally, constraints (4) ensure the non-negativity of the continuous variables x_{ij}^k and force the discrete variables y_{ij} to assume binary values.

Note that, although the network is undirected, the flow is directed, so once the decision of connecting two nodes i, j has been taken, flow in any direction will be allowed, that is, both arcs (i, j) and (j, i) will be considered in the design.

The objective is to minimize the overall cost incurred by designing and operating the network.

3. Getting initial solutions

The design and implementation of heuristics requires an efficient computational representation of solutions. This requirement is accomplished here by splitting a solution into independent elements, one element by commodity, called *blocks of paths*. A block of paths for a given commodity k will be referred as B^k. Then, for each commodity k whose demand will be transported on the network, a block B^k, formed by one or more paths selected from a list of shortest paths, is generated.

Even though commodities may share the capacity of the edges, the blocks of paths are considered independent from each other.

3.1. Generation of shortest paths

The method for finding a solution begins with the generation of paths. Therefore, q shortest paths between each origin-destination pair in the network are found, where q is a parameter given during implementation. In order to get q shortest paths, it is necessary to define what is meant by length of an arc (i, j). Three different types of arc lengths were considered, and in order to assign the same importance to each type of length, $q/3$ different paths were obtained for each one of those different lengths:

$$length_{ij}^k = \left(c_{ij}^k + F_{ij}\right)\left(1 + \left|\frac{(u_{ij} - d^k)}{d^k}\right|\right) \tag{5}$$

$$length_{ij}^k = \left(c_{ij}^k + F_{ij}\right)\left(1 + \left|\frac{u_{ij} - \sum_{h\in\Psi_{ij}} d^h}{\sum_{h\in\Psi_{ij}} d^h}\right|\right) \tag{6}$$

$$length_{ij}^k = \left(c_{ij}^k + F_{ij}\right)\left(1 + \frac{f_{ij}}{f'}\right) \tag{7}$$

In (5) a penalty is attached to the deviation of the demand of commodity k that may be potentially transported through the arc, from the capacity of the edge corresponding to that arc.

In (6) Ψ_{ij} is the subset of commodities which share edge $\{i, j\}$ (that is arc (i, j) or (j, i)) in those paths found with type 1 length. In this case, a penalty is attached to the deviation of the sum of demands of those commodities that may be potentially transported through the arc, from the capacity of the edge corresponding to that arc.

Finally, in (7) a penalization factor is considered to introduce some diversification in the set of paths being generated. f_{ij} stands for the frequency of edge $\{i, j\}$ in the previously

found paths, i.e., how many times edge $\{i, j\}$ has appeared in some other paths, even if they belong to some other commodity. f' stands for the maximum frequency of appearance of an edge in the previously constructed paths.

It is worth noticing that the length of an arc may vary from commodity to commodity. Paths obtained with type 1 and 2 lengths are expected to be good, while paths obtained with the third type are more diversified when compared to the others.

Note that for each commodity being transported over the network, q shortest paths must be generated. This set of shortest paths will be denoted LP^k.

An implementation of Lawler's algorithm described in Lawler (1972) was used to generate the q shortest paths.

3.2. GRASP generalities

In order to obtain initial solutions for the problem, a heuristic based on GRASP was designed. GRASP was developed by Feo and Resende (1989) to study a high complexity covering problem. GRASP is a multi-start method, and each GRASP iteration randomly constructs a greedy solution followed by a local search using that solution as an initial starting point. Thus, GRASP consists of two phases: a solution construction phase and a local search to improve the solution.

As it is the case in most GRASP implementations (Argüello, Bard, and Gang, 1997; Fernández and Martí, 1999), the constructive phase has two main features: adaptive greedy measures, and random selection. Later, adaptive memory features have been included, as proposed by Fleurent and Glover (1999), to retain and analyze the characteristics of selected solutions and so providing a base for improving later executions of the constructive process.

A trial solution is formed in the constructive phase and then it is improved by means of a post processing mechanism until a local optimum is reached. The constructive mechanism produces a solution incorporating one element (i.e. a *path*) at a time, so in each step of the process, there is at hand a partial solution. An element that can be selected as part of a partially constructed solution is called a *candidate element*. To determine which element will be selected to be included in a partial solution a greedy function is used. A greedy function measures the local contribution of each candidate element to the partial solution. A greedy election consists in choosing the candidate element with the best value of its greedy function. In order to introduce some randomness to this procedure a *restricted candidate list* (RCL) is used for each commodity. This list, from which an element is randomly selected, is formed by high quality elements, that is, candidate elements with the best values of their greedy function. The length of this list can be fixed or dynamic.

Once an element has been added to the partially constructed solution, the values of the greedy function must be reevaluated. This makes the procedure acquire the *adaptivity* feature that characterizes it.

A local search algorithm explores repeatedly the neighborhood of a newly constructed solution in order to find a better solution. If a better solution is not found it is said that a local optimum has been reached. The local search plays an important role in GRASP as it tries to look for locally optimum solutions in promising regions of the solution space.

3.2.1. Greedy objective. In order to measure the benefit of incorporating a path p to s^*, the solution being created, a function *value*(s^*,p) is defined. This function is defined in such a way that greater values correspond to better choices.

To define the function *value* other measures must first be established. Let $C(B^k, p)$ be the function that represents the cost of assigning path p to B^k:

$$C(B^k, p) = \sum_{(i,j)\in p} \left(c_{ij}^k d'^k + F'_{ij}\right)\left(1 + \left|\frac{u'_{ij} - d'^k}{d'^k}\right|\right) \tag{8}$$

where p is any path from LP^k; F'_{ij} equals F_{ij} if edge $\{i, j\}$ has not yet been used, or 0 if it already has. u'_{ij} is the residual capacity of edge $\{i, j\}$; d'^k is the residual demand for commodity k (that is, the portion of the demand for commodity k waiting to be transported); and c_{ij}^k, is the variable cost per unit of flow of commodity k using arc (i,j).

It is worth noticing that F'_{ij} represents the cost of using the edge $\{i,j\}$ for the first time this edge appears in the network design, or, it is set to zero when the edge has been already included, that is, edge {i,j} has appeared in some path already included.

The cost of this partially constructed solution s^* if path p is assigned to B^k can be obtained as follows:

$$T(s^*, p) = \sum_{m=1}^{k-1} \sum_{l\in G^m} C(B^m, l) + \sum_{p\in G^k} C(B^k, p) \tag{9}$$

where G^m is the set of paths contained in each B^m. This function separates the paths belonging to previously constructed blocks from the block of the commodity being analyzed, that is, commodity k. This is done because function T will be used to evaluate the set of candidate paths for the commodity k.

Define now *best_cost*(LP^k) = min $\{T\ (s^*,l) : l \in \mathrm{LP}^k\}$ as the minimal cost in that list. The cost of each path is normalized considering *best_cost*(LP^k) in such a way that the value of the function defined for each p belonging to that list is calculated as:

$$value(s^*, p) = best\_cost(\mathrm{LP}^k)/T(s^*, p) \tag{10}$$

It can be noticed that function *value*(s^*, p) will take values in the range [0,1].

3.2.2. Construction of RCL^k and selection of paths from LP^k. To form the restricted candidate list for commodity k, RCL^k, the set of paths for commodity k is rearranged in descending order according to the function *value*(s^*, p). The length of RCL^k will vary dynamically as feasible solutions are being created or as the GRASP advances in iterations. To achieve this, an initial length is assigned to the RCL^k as a proportion μ (see Section 5.2.1) of the total number of paths for the commodity being analyzed. The RCL^k will grow gradually until its cardinality equals the total number of paths for a given commodity in the following way:

$$car\_RCL = \begin{cases} \mu q & \text{If } \mu \geq a(grasp\_iter, sol) \\ a(grasp\_iter, sol)q & \text{If } \mu < a(grasp\_iter, sol) \end{cases} \tag{11}$$

In this function $a(grasp\_iter, sol)$ represents the progress of the algorithm and it can be defined as follows:

$$a(grasp\_iter, sol) = \text{Max}\left\{\frac{grasp\_iter}{GRASP\_ITER}, \frac{sol}{MAX\_SOL}\right\} \tag{12}$$

where *graspiter* is the number of elapsed GRASP iterations, *sol* is the number of generated solutions at a given iteration; *GRASP_ITER* is the maximum number of GRASP iterations; and *MAX_SOL* is maximum number of solutions in a population.

car_RCL is used to obtain the length of RCL^k. If the procedure has iterated several times without constructing solutions, then it is assumed that a demand for a given commodity k could not be sent with the paths in RCL^k, therefore, the length of RCL^k must be increased. On the other hand, if GRASP has generated a solution in each iteration, then we need to diversify including new paths in the list and therefore the length of RCL^k must be increased. Paths are randomly selected from this list.

Once a path is selected, the capacity of this path is then checked considering the arc with least residual capacity in the path. As much flow as possible will be sent using this path if and only if the path capacity is smaller than the residual demand for this commodity, and other path should be selected. Otherwise, only the necessary flow to complete the demand will be sent and another commodity will be analyzed.

The adaptive feature in GRASP is guaranteed here by the continuous evaluation of paths. This means that every time a path selection has been made, the remaining paths have to be reevaluated and a new RCL^k for commodity k is constructed. In this case, as q grows, so does the time this procedure takes to analyze all paths, then it is important to consider, first, the quality of paths (then the importance of path length calculation), and second, the number of paths or q (which must be kept low).

3.2.3. Improvement routine. As explained, paths are selected from the RCL^k to become part of a block of paths for commodity k. The cost of this block of paths is defined as:

$$H(B^k) = \sum_{p \in B^k} \sum_{(i,j) \in p} \left(c_{ij}^k x_{ij}^k + F'_{ij}\right) \tag{13}$$

Once a block of paths is completed for a new solution being created, a routine called "improvement routine" is then executed. Basically, the improvement routine consists on sorting the paths to obtain a better distribution with lower cost, if this were possible. The following steps describe this procedure:

1. *Get a cost for each path*. For each p in B^k, let $ucost(p)$ be the cost of transporting one unit of demand through this path:

$$ucost(p) = \sum_{(i,j) \in p} \left(c_{ij}^k + F'_{ij}\right) \tag{14}$$

2. *Sorting*. Sort paths in block k in non decreasing order by (14).

3. *Create a new block of paths*. Select a path from the sorted list of paths. Send as much flow as possible. If flow sent through this path is greater than zero, add path to new block of paths. Proceed until all the demand has been sent.
4. *Get the cost of a new block of paths*. If a new block of paths was created then let $H'(B^k)$ be the cost of that block.
5. *Choose the best block of paths*. If $H'(B^k) < H(B^k)$, delete the previous block of paths, keep the new one; otherwise delete the new block of paths and keep the previous block.

3.3. Incorporating memory into GRASP

At this point, every GRASP construction is made independently of each other, in other words, there is no learning involved in this process. In order to learn from previously generated solutions, and to obtain useful information that will help in generating future solutions, the concepts of strongly determined variables and consistent variables were implemented. Strongly determined variables depend on a set of elite solutions whose values cannot change without substantially degrading the value of the objective function. A consistent variable is one that receives a particular value in a significant portion of good solutions. As proposed by Fleurent and Glover (1999), a set S of r elite solutions is maintained and it is updated (if proceeds) each time a new solution is generated. As new feasible solutions are being generated, these may replace those in S.

3.3.1. Evaluation of paths to build solutions relative to S. In order to learn from previously generated solutions, and from previously selected paths, it is necessary to make use of a function that evaluates the benefit of incorporating a path in a block of paths for a given commodity. As defined before, let $value(s^*, p)$ be the function which evaluates the benefit of including path p in s^*, the solution being created. Additionally, define a new function which may retain the characteristics of previously generated elite solutions. This function is such that it measures the strongly determined and consistent features of selecting path p to be included in s^*. Let $intensity(s^*, p)$ be such a function. Larger values of this function mean that this choice has occurred more frequently in the best members of S.

An evaluation that takes into account both value and intensity of a choice may be defined as a monotone increasing function of its arguments as follows:

$$E(s^*, p) = F(value(s^*, p), intensity(s^*, p)) \tag{15}$$

where $value(s^*, p)$ is defined in (10). Consider now the set S of elite solutions. Let $Cost(s)$ be the cost of a solution measured by the value of its objective function. Define $best\_cost(S) = \min\{Cost(s): s \in S\}$. The cost of each solution in set S is then normalized in the same way that was done for paths. Therefore, a function *Value* is defined for each solution in S as follows:

$$Value(s) = best\_cost(S)/Cost(s) \tag{16}$$

This value represents a crude measure of the "strength" of paths in s. Now, define the intensity function of assigning path p to the new solution s^*. This function will take into account the frequency of this path in set S.

$$intensity(s^*, p) = \sum_{\{s \in S | p \in B^k\}} Value(s) \tag{17}$$

A simple way of defining function $E(s^*, p)$ as an increasing function of its arguments is shown:

$$E(s^*, p) = \lambda value(s^*, p) + intensity(s^*, p) \tag{18}$$

Different values of λ will cause more emphasis on the diversity or on the intensity term which in turn will guide the selection of paths. Low values of λ, even zero, will cause more emphasis on the intensity allowing the selection of paths that have appeared in high quality solutions. High values of λ will cause the selection of paths in a greedy fashion, that is, paths highly evaluated regardless of their contribution in the set of elite solutions.

Note that $intensity(s^*, p)$ will take values in the range $[0, |S|]$.

The value of λ will vary dynamically as the diversity of population grows. In order to achieve this, a measure for the diversity of population is assessed. To obtain this measure, all 2-elements subsets from the population of solutions generated by GRASP are formed and for each subset the distance between the two solutions is computed. This distance is then averaged and compared to a certain threshold. If this distance exceeds this threshold, λ is decreased, otherwise it is increased. More details in Section 5.2.1.

3.3.2. Construction of RCL^k and path selection from RCL^k. Paths in LP^k are ordered according to their evaluation function (18). A fixed length for the RCL^k is considered now and only those best evaluated paths are included. Best evaluated paths according to function (18) are frequently used paths and/or low cost paths according to function (10). For this case, experiments showed that a fixed length for RCL^k is more convenient due to the fact that this list will now contain only the best paths, and allowing the procedure to select from larger lists will only cause selection of bad paths. Several experiments to determine this length were carried out and best results were obtained with smaller lists, in this case a 10% of q.

In addition to what has been said, and in order to give the best evaluated paths a greater opportunity of being selected, $E(s^*, p)$ will be mapped into positive values over RCL^k. Then the probability of selecting path p from RCL^k, denoted $\Pi(s^*, p)$, is established to be strongly biased toward choosing the members of RCL^k with larger $E(s^*, p)$ values.

For each path in RCL^k, the probability of being selected is defined as follows:

$$\prod(s^*, p) = \frac{E(s^*, p)}{\sum_{l \in RCL^k} E(s^*, l)} \tag{19}$$

3.3.3. Updating set S of elite solutions. The constructive process will be guided by using the set S of elite solutions. Initially, S contains r "null" solutions with infinite cost, where

"r" is a parameter given during the implementation phase. First "r" generated solutions by GRASP take place in S. From that moment on, any solution being created will replace some other in S if the following is achieved: it is better than the best solution in S or else, it is better than the worst solution in S but distant enough from the rest of solutions in S.

The distance between two solutions considers how many different paths can be found in each of the solutions. Then, a function δ that measures this distance is defined:

$$\delta(s^1, s^2) = \frac{\sum_{k \in K} \sum_{p_j \in LP^k} \left| t_j^1 - t_j^2 \right|}{h_1 + h_2} \tag{20}$$

where $\mathbf{t}^i$ is the characteristic vector of solution s^i corresponding to the paths (taken from LP^k) included in the block of paths for each commodity k. h_i is the number of paths used in solution i. Therefore, $t_j^i = 1$ if solution s^iuses path p_j and $t_j^i = 0$ otherwise. Then, vector $\mathbf{t}^i$ has $q|\mathrm{K}|$ components. This function will be zero if both solutions are identical, will be 1 if they are completely different (they don't share any path), and it takes a value between 0 and 1 in any other case.

To determine if a newly created solution $s*$ could gain access to elite set S, its objective function value will be first obtained; if this value is better than the best objective value in S then it gains membership to this elite set and the solution with the worst objective function value will be eliminated from S; on the other hand, if the objective function value was better (lower) than the worst in S, the distance δ of $s*$ to every other solution in elite should be calculated and averaged. This average has to be greater than certain threshold (η) to replace the worst solution in S.

4. Evaluation of initial solutions with scatter search

From the standpoint of metaheuristic classification, Scatter Search (SS) may be regarded as an evolutionary (or also called population-based) algorithm that constructs solutions by combining others. Unlike other evolutionary methods, Scatter Search has a structured and intelligent way of combining solutions, a method for updating a reference set of high quality solutions and diverse solutions and mechanisms for improving the newly created solutions. All these features distinguish SS from other methodologies also based on populations and make it attractive to tackle the problem addressed here.

As described in tutorial articles (Glover, 1998; Laguna, 2002) and other implementations based on this framework, the methodology includes the following basic elements:

- Generation of a population P
- Extraction of a reference set R
- Combination of elements from R and update of R

The dimension and structure of the solution set in different evolutionary algorithms may vary. Genetic algorithms, for example, work with the whole created population (typically 100 solutions), memetic algorithms work with small (and sometimes structured) population

(Moscato, 2000), while SS works with a subset of 10 to 20 solutions from the set of created solution. This subset called reference set R is built from the population P (generated by the diversification generation method) with only a few solutions from P. The way the reference set is initialized, updated and rebuilt is a crucial aspect in Scatter Search performance. If the construction of a reference set was made only based on the solution quality, the reference set would be formed by selecting the best b solutions in P. Nevertheless, a desired characteristic in general search procedures, and particularly in SS, is an adequate balance between intensification and diversification.

Next step in the Scatter Search methodology is the combination of elements in the reference set. To accomplish this, two or more elements from R are chosen in a systematic way with the purpose of creating new solutions. This is achieved by the construction of certain subsets of solutions from R and by applying the combination method to solutions in each one of these subsets. This combination is intended to be intelligent so a better solution than those in the subset may be created. At this point, it is possible that as a result of the combination method an infeasible solution be created. In this case, the combination method must have a procedure to restore feasibility. As new solutions are being created, these will gain membership to the reference set not only by their quality, but by their degree of diversity. The general procedure may iterate several times to achieve a better quality in the created solutions. The way SS combines solutions and updates the set of reference solutions used for combination sets this methodology apart from other population-based approaches.

The fact that these mechanisms of SS are not restricted to a unique design allows the exploration of strategic possibilities that may be effective in a particular implementation. These observations and principles lead to the following template presented by Fred Glover (1998), which consists of the following specific subroutines:

- A Diversification Generation Method.
- An Improvement Method.
- A Method for Generating a Reference Set.
- A Method for Generating a Subset.
- A Method for Combining Solutions.
- A Method for Updating Reference Set.

They will be described in the following sections.

4.1. A diversification generation method

This method is used to create a set of candidate solutions emphasizing systematic generation over randomization. This set is typically used to initialize the reference set and to rebuild it whenever it is required during the search phase.

In recent implementations of SS (Laguna and Armentano, 2004) the convenience of using frequency memory to develop effective diversification generation methods has been considered.

Taking into account what has been said, the diversification generation method implemented in this work to obtain an initial population P is based on the well-known technique called GRASP (Feo and Resende, 1995) explained in Section 3.

4.2. Improvement method

The objective of this method inside the SS structure is to transform a candidate solution into a candidate solution with better characteristics. In the original definition of the improvement method a feasible solution it is not required as input nor as output, though, it is usually expected that an output solution will be feasible.

Most improvement methods in the context of SS consist of a simple local search and typically they are applied to candidate solutions that were created with the generation diversification method or else with the method for combining solutions described later. Fundamental aspects relative to the improvement method are the frequency of its application and the deep of the associated search.

The implemented improvement method for MCND, as explained in Section 3.2.3, works as follows: each time GRASP generates a block of paths for a commodity in a solution, that block of paths is analyzed as to get, if this is possible, a new block with better cost. This procedure consists basically in obtaining a different flow distribution using the same paths rearranged in different order but with a lower transportation cost.

4.3. Reference set generation

Scatter Search doesn't work with the whole created population P, but it uses a small set of solutions called reference set (R).This set is built by selecting b solutions from P.Typically, the size of R is such that $b = 0.1|P|$. The selection of solutions that will be included in the reference set, considering their quality as well as their diversity, will have a high impact on the quality of new solutions generated by the combination method.

Several alternative criteria may be used to add solutions to the reference set. If the initialization of R was made exclusively considering the solution quality, the set will be built by selecting b best solutions from P.But if, on the other hand, a balance was desired between intensification and diversification, the set may be built by the union of two subsets R_1 and R_2 of sizes b_1 and b_2 respectively ($b = b_1 + b_2$), where R_1 would consider good solutions (as measured by their objective value), and R_2 would consider diverse solutions (Marti, Lourenço, and Laguna, 2000).

The quality and degree of diversification of the solutions have a major impact on the quality of the new solutions generated later by the combination method. In the present work, two different strategies for constructing the reference set were considered, then compared, and the results of this comparison shown in the computational results section. The first strategy, E_1(RS), was coupled to the diversification generation procedure based on a GRASP that does not include memory features , the second one, E_2(RS), was coupled to one that includes them. These two strategies are described in the next section. In addition, two different distance functions were tested to measure diversity. The

motivation for this is to gain insight on the impact of this function on the quality of solutions.

4.3.1. Strategy E_1(RS). The first strategy called E_1(RS) consisted in considering a reference set $R = R_1 \cup R_2$, with $|R_1| = b_1$ and $|R_2| = b_2$. In this way, R will be formed by $b = b_1 + b_2$ elements, where some of them (b_1) will be high quality solutions, and some other (b_2) will be diverse solutions respect to the first ones. This strategy has been successfully used in some applications (Martí, Lourenço, and Laguna, 2000).

The procedure to generate a reference set using this strategy is as follows:

1. Solutions in P are ordered according to their objective value.
2. b_1 best solutions are added to R.
3. For each solution y in $P \backslash R$, distance from y to every element in R is calculated.
4. Solution y^* from $P \backslash R$ with greater minimum distance is selected and added to R.
5. Solution y^* is deleted from P.If $|R| = b$ stop, else proceed to step 3.

Consider a function Δ that measures the distance in edges between two any solutions y^1, y^2 from P. Unlike function δ defined in (20), this function measures the distance between two solutions in edges instead of paths. Define function Δ:

$$\Delta(y^1, y^2) = \sum_{\{i,j\} \in E} \left| t_{ij}^1 - t_{ij}^2 \right| \tag{21}$$

In (21) $\boldsymbol{t}^1$ y $\boldsymbol{t}^2$ are the characteristic vectors of solutions y^1 y y^2 respectively, relatives to the edge set E, in such a way that if edge $\{i, j\} \in E$ is present in solution y^1 the value of t_{ij}^1 is equal to 1, and 0 otherwise; similarly for solution y^2.

The minimum distance $\Delta_{\min}(y)$ of a solution y in $P \setminus R$ is calculated as follows:

$$\Delta_{\min}(y) = \min_{y' \in R} \{\Delta(y, y')\} \tag{22}$$

4.3.2. Strategy E_2(RS). The second strategy E_2(RS) considered to form the reference set was coupled to the diversification generation method based on a GRASP including memory features.

As mentioned above, memory structures are included in order to learn from previously generated solutions and to obtain useful information to build subsequent solutions. Specifically an elite set S is created and updated (when necessary) each time a new solution is built. This new built solution will gain membership in S if its objective function value was better than the best solution in S, or, if it was better than the worst solution in S and distant enough with respect to other solutions in S.

Therefore, after the diversification generation method has created a population P, a set S of good (measured by their objective value) and at the same time relatively diverse solutions, is at hand.This set S will be the reference set considered in this strategy.

As the reference set considered in this strategy is the same elite set created by the diversification generation method, the distance function used to measure diversity is the function δ described in (20).

4.4. Subset generation

Solution combination methods in Scatter Search are not limited to combining only two solutions. That is why a subset generation method is needed in order to create subsets of different sizes, each subset containing two or more solutions to be combined. The methodology of Scatter Search is such that the set of all combined solutions (i.e., the set of all the solutions that the implementation will try to generate) may be produced entirely at the same time the subsets are created. Therefore, once a subset has been created, there is no need to create it again. This situation is completely different from the methodology of Genetic Algorithms where combinations are randomly determined.

The procedure of subset generation uses a strategy that expands two element subsets into subsets of greater size, controlling at the same time, the total number of subsets to be generated. In other words, the mechanism does not try to create all possible two element subsets, and then all possible 3 element subsets, and so on until it has reached the size of R. This technique would not be efficient as the number of subsets would grow drastically. Even in small subsets, combination of all possible subsets is not effective because a larger number of subsets would be almost identical and would produce the same local optima.

In this work, subsets type I, II, III, and IV introduced by Glover were used. This subsets have proved to be effective in several applications (Campos, Laguna, and Martí, 1999; Martí, Lourenço, and Laguna, 2000). The technique selects representative subsets of different size, generating 4 different types of subsets:

Type I Subsets: All 2 element subsets.
Type II Subsets: All 2 element subsets including best solution not considered in this subset.
Type III Subsets: All type II subsets including in each on best solution not considered in this subset.
Type IV Subsets: All subsets with best i elements for $i = 5, 6, \ldots, b$.

Considering a reference set of 10 solutions, there are at most 141 different subsets to combine.

This subset generation method is context-independent and there exist efficient implementations which avoid the duplication of subsets previously generated (Laguna and Marti, 2003).

4.5. Solution combination method

This method uses the generated subsets described in section 4.4 to combine the elements in each subset with the purpose of creating new solutions. The solution combination method is a problem related mechanism as it is deeply linked to the solution representation. Depending

on the specific way of combination, one or more solutions may be generated. In the present, the combination method generates a new solution (if possible) from a given subset.

Combination methods may be systematic or probabilistic. A systematic method is applied to each subset using invariable rules. Therefore a generated solution does not depend on probability, implying that this method will obtain always the same solution from the same subset. On the other hand, probabilistic methods assign probabilities which define an election to elements in a subset.

In this problem, each solution is formed, for each commodity k, by a block of paths which is capable to transport the whole demand for that commodity and that has an associated cost. It is important to note that this cost is dependent on and related to a given solution.

To build a new solution s_{new} from a given subset, the method will choose for each commodity k the block with the lower cost between all the solutions in that subset. At this point it is very likely to loose feasibility when trying to put together blocks from different solutions. That is why a routine specially designed to recover feasibility was developed.

The combination method may be outlined as follows:

- For each subset generated by the Subset Generation Method
  - For each commodity k
    - Select block of paths with lower cost from solutions in subset
    - Add this block to s_{new}
    - Apply recovery of feasibility mechanism if necessary

The cost of the block for this commodity k, relative to a solution s, is calculated using (13). It is worth noticing that this cost depends on the solution where this block is found, that is, a block with the same paths and flows in one solution will most probably have a completely different cost in any other solution.

The block with lowest $H(B^k)$ value, for every solution s belonging to subset under consideration will be added to the new solution s_{new}. Paths are added one at the time, verifying residual capacities for each one of their arcs. Residual capacities of arcs depend on flows in previously added blocks of paths to the new solution s_{new}. When possible, the original flow in each path is maintained. If the residual capacity of a path is lower than the capacity of the original block where the path comes from, as much flow as possible will be sent, and this block in s_{new} will not be able to transport the whole demand for commodity k.In this case, the recovery of feasibility procedure, explained ahead, will be applied.

4.6. Updating of the reference set

The reference set R is a collection of high quality solutions as well as diverse solutions used to generate new solutions by means of a combination method. The updating method for the reference set is used to maintain R.

New solutions created by the combination method will be evaluated to test their possibility to gain membership in the reference set. The performance of Scatter Search depends greatly on the strategies used to update the reference set (Laguna and Armentano, 2004). There are basically two strategies of updating the reference set: static and dynamic (Laguna and Martí, 2003).

The static updating works as follows: solutions which are built by the combination method are placed in a pool of solutions called L. After applying the combination method, the pool is full and the reference set is updated. The new reference set consists of best b solutions in $R \cup L$.

Alternatively, the reference set may be updated dynamically. The dynamic updating consists on evaluating the new combined solutions to gain membership in R as they are being created. In other words, instead of waiting until all possible combinations are made to update R (as it is the case in the static updating), if a new solution gains membership in the reference set, this last is immediately updated before next combination is carried out.

Even when dynamic updating is more aggressive in terms of the fast incorporation of high quality solutions to R, static updating was tested too because it guarantees that every solution in R is used at least once by the combination method.

Updating of R when it was created by strategy E_1(RS) proceeds in the following manner: Let s_{new} be a new feasible solution to MCND, candidate to replace another solution in R, let s_{worst} the worst solution in R_1,that is the solution with highest value of its objective function and let s_{dist} the solution in R_2 with smaller distance to R. The new solution s_{new} will replace s_{worst} if $z(s_{new}) < z(s_{worst})$, or else, will replace s_{dist} if the distance of s_{new} to the reference set R is larger than the distance of s_{dist} to the same set.

On the other hand, when R was created by strategy E_2(RS), the same mechanism used to update the elite set will be used. That is, a newly created solution s_{new} may replace s_{worst}, the solution with the highest value of its objective function in R if $z(s_{new}) < z(s_{worst})$ and s_{new} is sufficiently different from the elements in R. The distance or proximity of s_{new} to other solution in R is measured by averaging the distance between s_{new} and every other solution in R. If this average is less than a certain threshold, solution is discarded, unless an aspiration criterion is satisfied, in this case, the objective function value of s_{new} is smaller than any other objective function value of a solution in R.

Considering what has been said, several experiments were carried out. First, two Scatter Search strategies using distance functions Δ and δ with dynamic update and E_1(RS) were tested. Function Δ performed better than function δ $SS^{(rde)}$) so this procedure was used for future comparisons. Later, different strategies (dynamic and static) for updating the reference set as well as different distance functions (δ and Δ) were tested. In this case results are shown on different tables in Section 5 under $SS^{(edp)}$, $SS^{(ede)}$, $SS^{(esp)}$ and $SS^{(ese)}$ columns as shown below:

	$SS^{(rde)}$	$SS^{(edp)}$	$SS^{(ede)}$	$SS^{(esp)}$	$SS^{(ese)}$
Generation of R	E_1(RS)	E_2(RS)	E_2(RS)	E_2(RS)	E_2(RS)
Updating of R	dynamic	dynamic	dynamic	static	static
Distance function	Δ(edges)	δ (paths)	Δ(edges)	δ (paths)	Δ(edges)

4.7. Feasibility recovery method

The application of the combination method to solutions in the reference set carries the possibility of generating an infeasible solution. Infeasibility is detected during the combination

procedure and before a new solution is completely generated. In order to recover feasibility for a partially generated solution a procedure of recovery of feasibility is applied. This method will try to recover feasibility but in case of failing to do so, the new solution will be discarded.

The feasibility recovery procedure proceeds in the following way: For each path in the infeasible block of paths an exchange of paths is considered. The candidate paths are taken from LP^k, the list of q shortest paths for commodity k. This exchange of paths takes one path out of the infeasible block of paths and introduces another path into it. The chosen path must be able to transport the demand of the leaving path plus the demand not transported in the original infeasible block of paths. As exchanges are considered, costs of entering paths are compared in order to select the best possible path whose inclusion in the block does not increase the cost of the original selected block of paths. As mentioned before, if the feasibility recovery procedure is not successful, then the new solution s_{new} is discarded.

5. Computational results

In order to gain insight into the performance of the proposed method, the best solution found by the heuristic was compared to the best integer solution found by CPLEX optimizer (ILOG CPLEX 6.6, 1999). All experiments were conducted on a SUN™ Ultra 10 computer. The code was compiled using Sun-C compiler. In each case, CPLEX was allowed to run for 6 hours. In most cases, it could not find the optimal solution.

5.1. Network instance generator

A random instance generator was designed to generate instances classified according to the following:

Class I	High Fixed Costs, Loosely Capacitated Networks
Class II	High Fixed Costs, Tightly Capacitated Networks
Class III	High Variable Costs, Loosely Capacitated Networks
Class IV	High Variable Costs, Tightly Capacitated Networks

Different sizes of network instances were considered in this classification: 30 nodes and 350 edges (700 arcs), 50 nodes and 700 edges (1400 arcs). For each size classification, 10, 50, and 100 commodities were tested. Generated instances are available at http://yalma.fime.uanl.mx/∼karim

Table 1 shows a total of 120 generated instances. For each group of 10 generated instances, 5 instances were generated with 30 nodes and 350 edges and the other 5, with 50 nodes and 700 edges.

Loosely and tightly capacitated networks were obtained applying the following capacity ratio:

$$\rho = \frac{\sum_k d^k MF^k}{\left(\sum_k d^k\right)^2} \tag{23}$$

Table 1. Generated instances.

	Commodities			
Classification	30	50	100	Total
Class I	10	10	10	30
Class II	10	10	10	30
Class III	10	10	10	30
Class IV	10	10	10	30
Total	40	40	40	120

where MF^k is the maximum flow between O(k) and D(k). When ρ takes values less than 1, instance is considered tight, and when ρ increases beyond 1 the network becomes looser. The rationale behind this ratio is the fact that a feasible network must satisfy the following relation:

$$MF^k \geq d^k \forall k \in K \tag{24}$$

Then, for a feasible network, it follows that $(\Sigma_{k\in K} MF^k) / (\Sigma_{k\in K} d^k) \geq 1$. Nevertheless, as interested in measuring the tightness of a feasible network, this ratio should be scaled close to 1 for tight networks. Consequently, for a given commodity *k*, MF^k was multiplied by the relative importance of the demand of commodity *k* over the whole demand in the network, that is, $d^k/(\Sigma_{k\in K} d^k)$. As this ratio grows farther than one, the network is considered less tight.

Instances were adjusted to ensure feasible solutions. This was done by letting CPLEX obtain a feasible solution, when CPLEX detected infeasibility then the capacities were scaled to gain feasibility. Fixed costs, for instances where these costs were considered relevant, were obtained as follows:

$$F_{ij} = \frac{\sum_{k\in K} C_{ij}^k}{|K|} \tag{25}$$

This value was scaled down to 20% in order to get instances with relevant variable costs. In order to get instances classified the following parameters were used:

Demand for commodity k: Uniformly distributed between 1 and 100.

Variable costs: Uniformly distributed between 1 and 100. A different variable cost was generated for each commodity and each edge.

Edge capacity: Uniformly distributed between 0.8 and 1.2 of total demand (sum of commodity demands) on the network. This value was then scaled in order to get tighter capacities.

Fixed costs: Fixed cost values were obtained multiplying the average variable cost by the edge capacity. This value was then scaled in order to give predominance of fixed costs over variable costs.

5.2. *Experimentation and analysis*

5.2.1. Parameters. It is well known that most heuristic procedures depend highly on their parameter values. GRASP and Scatter Search are no exception to this rule. Several parameters have been analyzed and tested to check for their relevance and the following results were obtained.

grasp_iter	1000	This number represents the iterations that GRASP was allowed to run in order to build a population of 100 solutions.
λ^o	10	Initial value of λ. Used in GRASP with memory to give emphasis to value function. This number indicates that, initially, it is desired to get good solutions, as measured by their objective value. Then, it will be decreased as to get intensified solutions.
$\Delta^1(\lambda)$	5	Decrement of λ. Different experiments were carried out to determine the value of this parameter. When population is diverse, a fast intensification is desired.
$\Delta^2(\lambda)$	3	Increment of λ. Experiments showed that the increment of λ should be slower than its decrement.
ε	0.35	Diversity threshold for populations in GRASP with memory. Several values of ε varying from 0.1 to 0.75 were tested. As this value is essential for increasing and decreasing λ, if it is set too high, λ will rarely be changed, that is why it was set to a lower value.
η	0.30	Substitution threshold for elite set used in GRASP with memory. Each time a solution is generated by this procedure, it is tested for membership in the Elite set of good and diverse solutions. Experiments varying η from 0.1 to 0.45 were carried out. Best results were obtained with $\eta = 0.30$
$q^{(1)}$	30	Number of shortest paths for memoryless GRASP. Several values of $q^{(1)}$ were tested, but in general good results were obtained using $q^{(1)} = 30$. As this number increases solution quality does not necessarily increases, but execution time does. A compromise between quality of solutions and execution time was the goal when changing this parameter.
$q^{(2)}$	50	Number of shortest paths for memory GRASP. As $q^{(2)}$ grows, so does the quality of solutions in GRASP as well as execution time. Several values of q were tested, but in general good results were obtained using $q^{(2)} = 50$. As this number increases, so does the time the heuristic takes in constructing solutions, therefore we tried to keep it as low as possible.
μ	0.1	Proportion of q. Used when assessing the length of the restricted candidate list for the memoryless GRASP. Several values of this parameter were tested, varying from 0.05 to 0.35. Best results were obtained using $\mu = 0.10$.
scat_iter	14	Number or populations Scatter Search will analyze. For each one of these iterations, a population of 100 solutions must be generated, then increasing this number will degrade the overall execution time of the procedure, but on the other hand, will normally increase solution quality.
η'	0.35	Substitution threshold for reference set in SS. This parameter is used in the last four SS strategies (those using a GRASP with memory) for gaining membership to the reference set. Values varying from 0.15 to 0.45 were tested. Better results were obtained with 0.35.

Table 2. Comparison between SS procedures grouped by proposed classification.

Classification	$SS^{(rde)}$ (%)	$SS^{(edp)}$ (%)	$SS^{(ede)}$ (%)	$SS^{(esp)}$ (%)	$SS^{(ese)}$ (%)
Class I	4.25	0.97	0.86	0.89	0.85
Class II	17.18	8.05	8.29	8.31	8.44
Class III	6.82	0.13	−0.77	−0.23	−0.46
Class IV	5.51	2.15	1.93	2.02	2.06
General	8.86	2.97	2.72	2.90	2.87

5.2.2. Comparison between different scatter search strategies. The five different Scatter Search strategies were compared and the results were compared against CPLEX and reported in tables shown below. The different strategies are: $SS^{(rde)}$, $SS^{(edp)}$, $SS^{(ede)}$, $SS^{(esp)}$ and $SS^{(ese)}$. For each one of them, the diversification generation method was called 14 times, that is, 14 different populations of 100 solutions each one were generated.

In Table 2 instances are presented by the proposed classification (cost and capacity). First column shows this classification. Class I: High Fixed Cost and Loosely Capacitated Networks; Class II: High Fixed Cost and Tightly Capacitated Networks, and so on. Columns 2, 3, 4, 5 and 6 show the average relative gap between SS procedures when compared to CPLEX best integer solution. It is easy seen from results in Table 2 that a Scatter Search procedures with embedded memory features over perform the traditional Scatter Search, that is, the one without memory features. As for the Scatter Search procedures using different ways of updating the reference set and using different distance functions, it can be seen that results are quite similar, nevertheless the best results were obtained with $SS^{(ede)}$ procedure.

It is worth noticing that results in Table 2 show that a bad network design is highly penalized when fixed costs are relevant, therefore the gap between CPLEX and SS is greater. Loosely capacitated networks with high variable costs are easier to solve and, in general, Scatter Search results for these kinds of networks are better than those of CPLEX (showed by negative values in Table 2).

The predominance in most situations of $SS^{(ede)}$ over the other SS strategies led us to observe the execution time for the different SS strategies. It took (on average) 79 seconds for $SS^{(esp)}$ to process an instance, while $SS^{(ede)}$ took 151 seconds on average. Results show that additional computational time taken to verify distance between solutions using edges instead of paths is not worthy enough.

Inclusion of memory features does not degrade the performance of the proposed heuristic. Focusing on results for $SS^{(esp)}$ it can be noticed that even when computation time increased 27%, solution quality increased 67% with respect to the Scatter Search procedure without memory features ($SS^{(rde)}$). Results on execution time are shown in Table 3.

Even when the results for the proposed classification show the cost-capacity interaction, it is interesting to see how other parameters affect the performance of the heuristic. So, analysis for different groupings of the same SS results allowed drawing the following conclusions:

Table 3. Execution time in seconds for the 5 different SS procedures.

Classification	$SS^{(rde)}$	$SS^{(edp)}$	$SS^{(ede)}$	$SS^{(esp)}$	$SS^{(ese)}$
Class I	67	87	143	80	181
Class II	54	71	143	74	114
Class III	71	94	163	86	211
Class IV	56	73	156	76	126
General	62	81	151	79	158

Table 4. Performance of $SS^{(edp)}$, $SS^{(ede)}$, $SS^{(esp)}$, $SS^{(ese)}$ over $SS^{(rde)}$.

Nodes-Comm.	$SS^{(edp)}$ (%)	sec (%)	$SS^{(ede)}$ (%)	sec (%)	$SS^{(esp)}$ (%)	sec (%)	$SS^{(ese)}$ (%)	sec (%)
50–50	−0.13	88	−2.76	206	−2.81	85	−2.67	227
50–100	−3.86	149	−5.58	355	−5.40	154	−5.68	272
General	−1.99	118	−4.17	281	−4.10	120	−4.18	249

- As number of commodities grows, so does the difference between SS and CPLEX, leading to the conclusion that the number of commodities affects the heuristic procedure performance.
- Networks with fewer commodities are easier to solve, and this is reflected in the quality of solutions obtained by the heuristic procedure.
- Solutions loose quality (even for CPLEX best integer solutions) when fixed costs are high. This leads to the conclusion that errors when designing the network (that is, choosing wrong edges) will result in highly cost-penalized solutions (bad solutions).

For instances where a comparison between CPLEX feasible solution and SS solutions was not possible, strategies $SS^{(edp)}$, $SS^{(ede)}$, $SS^{(esp)}$, $SS^{(ese)}$ were compared to $SS^{(rde)}$. The result of this comparison is shown in Table 4. Column 1 shows the networks for which CPLEX could not find a feasible solution. Columns 2, 4, 6 and 8 show the average relative gap between SS procedures when compared to $SS^{(rde)}$. The other columns show average time in seconds for SS. Negative numbers in this table show the percentage that these strategies outperformed $SS^{(rde)}$. As it is easly seen in this table, in general, $SS^{(ede)}$ is better than the rest of the SS strategies, but again, processing time is much higher.

5.2.3. Comparison between scatter search and dual ascent procedure. In general, the Scatter Search procedure has proved to be effective when solving network instances. Experiments carried out and reported in (Alvarez, De Alba, and González-Velarde, 2001) showed this when comparing results from strategy $SS^{(rde)}$ to Herrmanns's Dual Ascent method (Herrmann et al., 1996). In order to perform this comparison, network grid instances were generated reproducing Herrmann's experiment. Scatter Search procedure outperformed Dual Ascent method in all cases, as shown in Table 5.

Table 5. Comparison between Herrmann's dual ascent and scatter search.

Number of edges	Instances	Mean (%)	Std. Dev. (%)	Dual-Ascent (%)
22	56	0.402	0.488	3.7
31	56	0.627	0.526	4.5
40	56	0.600	0.549	–
52	56	0.753	0.605	–
60	56	0.915	0.654	–

Column 1 shows the number of edges (undirected arcs) contained in the grid. Column 2 shows the number of instances that were generated. Columns 3 and 4 show the arithmetic mean and the standard deviation when the SS results were compared to CPLEX results. Column 5 shows the difference between the results obtained in Herrmann et al. (1996) using the Dual Ascent approach and the optimum value (hyphens represent no optimum value obtained from the optimizer). Note that results in Table 5 show the relative gap between optimal solution and SS solution in a range varying from 0.4% to 1%, much lower than those obtained with the dual ascent procedure. Additional results on this experiment can be found in Alvarez, De Alba, and González-Velarde (2001).

6. Conclusion

In the present paper, a GRASP was embedded into a Scatter Search framework in order to find good (in terms of quality) solutions to a fixed charge, multicommodity, capacitated network design problem. GRASP is used to generate a population of solutions as a starting point to the Scatter Search procedure. In order to get a better perspective of the population generation mechanism, two different GRASP strategies were tested: a traditional GRASP (that is, GRASP as reported in most implementations) and a memory based GRASP. Tests showed that even when memory based GRASP is slower, it outperforms the traditional implementation of GRASP.

As for the Scatter Search procedure, five different strategies have been developed and tested. These procedures were denoted $SS^{(rde)}$, $SS^{(edp)}$, $SS^{(ede)}$, $SS^{(esp)}$ and $SS^{(ese)}$ and were explained in detail in this paper. In general, procedures $SS^{(edp)}$, $SS^{(ede)}$, $SS^{(esp)}$ and $SS^{(ese)}$ over performed procedure $SS^{(rde)}$. The comparison was carried out against CPLEX best integer solutions. Instances used to test the heuristic were classified in an effort to gain a better perspective of performance. Results show that fixed costs are highly relevant when solving instances with this characteristic. When fixed costs are not very high, impact in results is not very relevant. The number of commodities tends to degrade the performance of heuristic in time as well as in solution quality, nevertheless fixed costs showed to be much more influential than any other characteristic of tested instances. In fact, high percentage deviations are expected as a slight variation from optimal design is highly penalized by fixed costs.

Results when solving instances showed that procedure $SS^{(ede)}$ is in general better than the rest of the SS procedures, nevertheless, extra effort spent in calculating distance of

solutions by edges is not worthy enough. Computing time is, on average, 43% higher for $SS^{(ede)}$ procedure than for $SS^{(esp)}$ procedure.

An important fact was drawn from tests of different q values(the number of paths between origin-destination nodes): much more important than the value of q, is the way the length of arcs is calculated. As the heuristic procedure relies heavily on *q-shortest* paths, if path lengths are not very accurately calculated, then the output of heuristic will reflect this miscalculation in lower quality solutions. This fact led us to work deeper on the way arc lengths should be calculated. Penalization procedures were used to avoid unattractive paths to be taken. Unattractive paths should be avoided especially for instances with high fixed costs. Although several values of q were tested, good results are in general obtained with $q = 50$ for the memoryless GRASP and $q = 30$ for the memory GRASP. For memoryless GRASP this value may be increased and, as expected, solutions are normally better, but extra computing time in evaluating paths increases considerably.

It is worth noticing that even when flow variables x_{ij}^k are defined continuous, the heuristic procedure gives integer flow values. This could be especially useful in case the original mathematical model would include the integrality condition on the flow variables.

There exist different applications based on this general model, as it is the case in Crainic (2000) and Gendron and Crainic (1996) which include another sets of restrictions as for example partial capacities on each commodity. In general, these restrictions may be included in the model presented here with small changes in the proposed algorithm.

Future works on this problem may include the application of exact methods for large scale problems, or the search for tight lower bounds as it is essential, specially for large instances, to have at hand a means for assessing the efficiency of an heuristic procedure.

Acknowledgments

This work has been partially supported by Conacyt under grant 36669-A and is part of a Research Chair in Industrial Engineering of ITESM titled "Extended Enterprises for Mass Customization". The authors wish to acknowledge the support of these grants in the preparation of this manuscript.

References

Alvarez, A., J.L. González-Velarde, and K. De Alba. (2001). "Scatter Search for the Multicommodity Capacited Network Design Problem." In *Proceeding of 6th Annual International Conference on Industrial Engineering–Theory, Aplications and Practice*, San Francisco, CA, USA.

Argüello, M.F., J.F. Bard, and G. Yu. (1997). "A Grasp for Aircraft Routing in Response to Groundings and Delays." *Journal of Combinatorial Optimization* 1(3), 211–228.

Arráiz, E., A. Martínez, O. Meza, and M. Ortega. (2001). "GRASP and Tabu Search Algorithms for Computing the Forwarding Index in a Graph." In *Proceedings of MIC 2001,* Porto, Portugal, July, pp. 367–370.

Balakrishnan, A. and T.L. Magnanti. (1989). "A Dual Ascent Procedure for Large-Scale Uncapacitated Network Design." *Operations Research* 37(5), 716–740.

Binato, S., W.J. Hery, D. Loewenstern, and M.G.C. Resende. (2002). "A Greedy Randomized Adaptive Search Procedure for Job Shop Scheduling." In Ribeiro, C. and Hansen, P. (eds.), *Essays and Surveys on Metaheuristics*, Kluwer, Boston, USA, pp. 58–79.

Campos, V., M. Laguna, and R. Martí. (1999). "Scatter Search for the Linear Ordering Problem." In D. Corne, M. Dorigo, and F. Glover (eds.), *New Ideas in Optimization*, McGraw-Hill: New York, USA, pp. 331–339.

Chardaire, P., G.P. McKeown, and J.A. Maki. (2001). "Application of GRASP to the Multiconstraint Knapsack Problem." In E.J.W. Boers, J. Gottlieb, P.L. Lanzi, R.E. Smith, S. Cagnoni, E. Hart, G.R. Raidl, and H. Tijink, (eds.), *Applications of Evolutionary Computing*, Lecture Notes in Computer Science, Vol. 2037, Springer, Heidelberg: Germany, pp. 30–39.

CPLEX Optimization, Inc. (1999). *ILOG CPLEX 7.1 Reference Manual*. Incline Village, NV: USA.

Crainic, T.G., A. Frangioni, and B. Gendron. (2001). "Bundle-Based Relaxation Methods for Multicommodity Capacitated Fixed Charge Network Design." *Discrete Applied Mathematics* 112(1–3), 73–79.

Crainic, T.G., M. Gendreau, and J. Farvolden. (2000). "A Simplex-Based Tabu Search Method for Capacitated Network Design." *INFORMS Journal on Computing* 12(3), 223–236.

Delgado, C., M. Laguna, and J. Pacheco. (2004). "Minimizing Labor Requirements in a Periodic Vehicle Loading Problem." To appear in *Computational Optimization and Applications*.

Feo, T.A. and M.G.C. Resende. (1989). "A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem." *Operations Research Letters* 8(2), 67–71.

Feo, T.A. and M.G.C. Resende. (1995). "Greedy Randomized Aaptive Search Procedures." *Journal of Global Optimization* 6(2), 109–133.

Fernández, E. and R. Martí. (1999). "GRASP for Seam Drawing in Mosaicking of Aerial Photographic Maps." *Journal of Heuristics* 5(1), 81–197.

Fleurent, C. and F. Glover. (1999) "Improved Constructive Multistart Strategies for the Quadratic Assignment Problem using Adaptive Memory." *INFORMS Journal on Computing* 11(2), 198–204.

Gendron, B. and T.G. Crainic. (1996). "Bounding Procedures for Multicommodity Capacitated Fixed Charge Network Design Problems." Publication CRT-96-06. Centre de Recherche sur le Transport, Université de Montréal, Montreal, Canada, January.

Gendron, B. and T.G. Crainic. (1994a). "Parallel Implementations of Bounding Procedures for Multicommodity Capacitated Network Design Problems." Publication CRT-94-45. Centre de Recherche sur le Transport, Université de Montréal, Montreal, Canada, September.

Gendron, B. and T.G. Crainic. (1994b). "Relaxations for Multicommodity Capacitated Network Design Problems." Publication CRT-965. Centre de Recherche sur le Transport, Université de Montréal, Montreal, Canada, February.

Glover, F. (1998). "A Template for Scatter search and path relinking." In J.K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, (eds.), *Artificial Evolution: Third European Conference*, Lecture Notes in Computer Science, Vol. 1363, Springer, Heidelberg, Germany, pp. 13–54.

Herrmann, J.W., G. Ioannou, I. Minis, J.M. y Proth. (1996). "A Dual Ascent Approach to the Fixed-Charge Capacitated Network Design Problem." *European Journal of Operational Research* 95(4), 476–490.

Hochbaum, D.S. and y Segev A. (1989). "Analysis of a Flow Problem with Fixed Charges." *Networks*, 19(3), 291–312.

Holmberg, K. and J. Hellstrand. (1998). "Solving the Uncapacitated Network Design Problem by a Lagrangean Heuristic and Branch-and-Bound." *Operations Research* 46(2), 247–258.

Holmberg, K. and D. Yuan. (2000). "A Lagrangian Heuristic Based Branch-and-Bound Approach for the Capacitated Network Design Problem." *Operations Research* 48(3), 461–481.

Johnson, D.S, J.K. Lenstra, H.G. y Rinnooy. (1978). "The Complexity of the Network Design Problem." *Networks* 8(4), 279–285.

Laguna M. (2002). "Scatter Search." In P.M. Pardalos, and M.G.C. Resende, (eds.), *Handbook of Applied Optimization*. pp. 183–193.

Laguna, M. and V.A. Armentano. (2004). "Lessons from Applying and Experimenting with Scatter Search." In C. Rego and A. Bahram, (eds.), *Adaptive Memory and Evolution: Tabu Search and Scatter Search*. Kluwer, (in press).

Laguna, M. and R. Martí. (2003). *Scatter Search Methodology and Implementations in C*. Kluwer, Boston, USA.

Lawler, E.L. (1972). "A Procedure for Computing the K Best Solutions to Discrete Optimization Problems and its Application to the Shortest Path Problem." *Management Science* 18, 401–405.

Magnanti, T., P. Mireault, and R. Wong. (1986). "Tailoring Benders Decomposition for Uncapacitated Network Design." *Mathematical Programming Study*, 26 112–154.

Magnanti, T. and R. Wong. (1984). "Network Design and Transportation Planning: Models and Algorithms." *Transportation Science* 18(1) 1–55.

Martí, R., H. Lourenço, and M. Laguna. (2000). "Assigning Proctors to Exams with Scatter Search." In M. Laguna and J.L. González-Velarde, (eds.), *Computing Tools for Modeling Optimization and Simulation: Interphases in Computer Science and Operations Research*. Kluwer: Boston, USA. pp. 215-227.

Moscato, P. (2000). Memetic Algorithms. In P.M. Pardalos and M.G.C. Resende, (eds.), *Handbook of Applied Optimization*. Oxford University Press, USA.

Pacheco, J.A. and S. Casado. (2004). "Solving Two Location Models with Few Facilities by Using a Hybrid Heuristic." A real health resource case. (In Press) *Computers and Operation Research No* 113.

Sridhar, V. and J.S. Park. (2000). "Benders-and-Cut Algorithm for Fixed-Charge Capacitated Network Design Problem." *European Journal of Operational Research* 125(3), 622–632.