An-Approach to Solve a Bi-Objective Un-Capacitated Facility Location Problem

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Computer Science & Engineering

By:
Bala Krishna Gudla
(Regn. No.800832028)

Under the supervision of:
Dr. Deepak Garg
Asst. Professor, CSED
&
Dr. Mahesh Kumar Sharma
Asst. Professor, SMCA

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2010

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"An Approach to Solve a Bi-Objective Un-Capacitated Facility Location Problem"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Computer Science and Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Dr. Deepak Garg and Dr. Mahesh Kumar Sharma, and refers other researcher's works which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

(Bala Krishna G)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Dr. Mahesh Kumar Sharma)
SMCA,
Thapar University,
Patiala

(Dr. Deepak Garg)
Computer Science and Engineering Department
Thapar University
Patiala

Countersigned by

12/07/10

(Dr. Rajesh Bhatia)
Head
Computer Science and Engineering, Department
Thapar University, Patiala

15.7.10

(Dr. R. K. Sharma)
Dean (Academic Affairs)
Thapar University,
Patiala

Acknowledgement

No volume of words is enough to express my gratitude towards my guide **Dr. Deepak Garg**, Department of Computer Science & Engineering and **Dr. Mahesh Kumar Sharma**, School of mathematics and computer applications Thapar University, Patiala, who has been very concerned and has aided for all the materials essential for the preparation of this thesis report. He has helped me to explore this vast topic in an organized manner and provided me with all the ideas on how to work towards a research-oriented venture.

I am also thankful to **Dr. Rajesh Bhatia,** Head of Department, D**r. Inderveer channa**, P.G. Coordinator, for the motivation and inspiration that triggered me for the thesis work.

I would also like to thank the staff members and my colleagues who were always there at the need of the hour and provided with all the help and facilities, which I required, for the completion of my thesis work.

Most importantly, I would like to thank my **parents** and the **Almighty** for showing me the right direction out of the blue, to help me stay calm in the oddest of the times and keep moving even at times when there was no hope.

G. Bala Krishna

G. Bala Krishna
(800832028)

ABSTRACT

Facility location, also known as location analysis, is a branch of operations research concerning itself with mathematical modeling and solution of problems concerning the placement of facilities in order to minimize transportation costs, avoid placing hazardous materials near housing, outperform competitor's facilities, etc. In a simple facility location problem, a single facility is to be placed, with the only optimization criterion being the minimization of the sum of distances from a given set of point sites. More complex problems considered in this discipline include the placement of multiple facilities, constraints on the locations of facilities, and more complex optimization criteria. Present thesis deals with a real life example of facility location problem which is basically a facility location problem which includes assignment of sites to the selected number of customers from among the given number of potential sites, for their requirements but with the objectives of minimizing cost for satisfying demands of all sites and minimizing the maximum time needed to fulfill the requirements of all the sites along with some constraints.

Thesis contains six chapters; first chapter is the introductory in nature and second chapter contains a brief review of literature related to this topic. In third and fourth chapter, un-capacitated facility location problem respectively. A heuristic approach consisting of a combination of add and drop rules incorporating NBHA (net benefit heuristic algorithm) and UFLTSA (un-capacitated facility location tabu search algorithm)has been used to find the set of efficient solutions.

Table Of Contents

List Of Figures

List Of Tables

Chapter1
Introduction

1.1Motivation

The single-objective model was the first to be developed and thus it was received considerably more exposure, been put to more use, and is generally considered to be a relatively high level of refinement. Thus, the implication is simple, well-tested tool is available and we may be inclined to fit the problem to this model despite the assumptions required. But in real life there are many problems with more than one objective for which the multi-objectives are required.

Dantzig's initial concept was centered about the development of a linear programming model but with a single objective. This so set the tone for the development of traditional linear programming that many (if not most) linear programming texts completely ignore even the possibility of more than one objective. Unlike the traditional single objective optimization problem where in it is settleon optimizing single objective function, there is no single universally accepted approach for solving the multi-objective optimization problems due to usually conflicting nature of objective functions leading to the situation where the optimization of one of these may adversely affect the optimization of others. So in the case of multi-objective optimization problems, there is no need to access the decision maker's utility function that may vary from decision maker to decision maker.

But in this approach is a bi-objective approach to solve a un-capacitated facility location problem using net benefit heuristic algorithm and tabu search algorithm. Here our proposed algorithm is optimizing the cost as well as time both are vice versa.

There are different methods to solve multi-objective optimization:

1. Weighting or utility methods

2. Ranking or prioritizing methods

3. Efficient solution (or generating) methods

1. Weighting or utility methods: The weighting method refers to those approaches that attempt to express all problem objectives in terms of a single measure. The basic

thrust of all such methods is to transform a multi-objective model in to a single objective model. As such, they are attractive from a strictly computational point of view (e.g., conventional simplex may be used if the model is linear). However, the obvious drawback to such an approach is that associated with actually developing truly credible weights.

2. The ranking or prioritizing methods: The ranking or prioritizing methods try to circumvent the heady problems, rather than attempting to find a numeral weight for each objective, they simply ask that objectives be ranked according to their perceived importance. Most decision makers can do this and, in fact ranking is a concept that seems inherent to much of decision making. For example, in determining raises, a procedure used by many organizations is to first rank one's employees in order of their work, productivity, value to the company, and so forth. The problem with the ranking approach is how to associate the results of given solution to the satisfaction of the ranking.

3. Efficient solution method: Efficient solution method "avoids" both the problems of finding weights and that of satisfying the ranking. It does this, or at least attempt to, by generating the total set of all efficient solutions (or non-dominated solutions, or Parto optimal solutions).

Modern problems tend to be very intricate and relate to analysis of large data sets. Even if an exact algorithm can be developed its time or space complexity may turn out unacceptable. But in reality it is often sufficient to find an approximate or partial solution. Such admission extends the set of techniques to cope with the problem. We discuss heuristic algorithms which suggest some approximations to the solution of optimization problems. In such problems the objective is to find the optimal of all possible solutions that minimizes or maximizes an objective function. The objective function is a function used to evaluate a quality of the generated solution. Many real world issues are easily stated as optimization problems. The collection of all possible solutions for a given problem can be regarded as a search space, and optimization algorithms, in their turn, are often referred to as search algorithms.

1.2 Thesis Organization

This thesis organized the thesis into 7 chapters which include Introduction; Background Information; Literature Review; Problem Statement; Proposed Algorithm; Computational Results; Conclusion. Chapter 1 describes the single objective model and methods of multi-objective methods and then follows by the whole thesis outline. Chapter 2, we discuss the back ground information related to tabu search algorithm. Chapter 3 we study the all about the un-capacitated facility location problem and capacitated facility location problem. Chapter4 discuss the problem statement and tasks. Chapter5 discuss the algorithm and sequence flow diagram, Chapter 6 computational results, and finally Chapter 7 summarizes the conclusions drawn in the thesis along with future research directions.

Chapter 2
Back Ground And General Concepts

2.1 Overview

The basic concept of Tabu Search as described by Glover (1986) is "a meta-heuristic superimposed on another heuristic. The overall approach is to avoid entrainment in cycles by forbidding or penalizing moves which take the solution, in the next iteration, to points in the solution space previously visited (hence "tabu"). The Tabu search is fairly new, Glover attributes its origin to about 1977 (see Glover, 1977). The method is still actively researched, and is continuing to evolve and improve. The Tabu method was partly motivated by the observation that human behavior appears to operate with a random element that leads to inconsistent behavior given similar circumstances.

As Glover points out, the resulting tendency to deviate from a charted course might be regretted as a source of error but can also prove to be source of gain. The Tabu method operates in this way with the exception that new courses are not chosen randomly. Instead the Tabu search proceeds according to the supposition that there is no point in accepting a new (poor) solution unless it is to avoid a path already investigated. This insures new regions of a problems solution space will be investigated in with the goal of avoiding local minima and ultimately finding the desired solution.

The Tabu search begins by marching to a local minima. To avoid retracing the steps used the method records recent moves in one or more Tabu lists. The original intent of the list was not to prevent a previous move from being repeated, but rather to insure it was not reversed. The Tabu lists are historical in nature and form the Tabu search memory. The role of the memory can change as the algorithm proceeds. At initialization the goal is make a coarse examination of the solution space, known as 'diversification', but as candidate locations are identified the search is more focused to produce local optimal solutions in a process of 'intensification'. In many cases the differences between the various implementations of the Tabu method have to do with

the size, variability, and adaptability of the Tabu memory to a particular problem domain. [1]

2.2 Tabu Search

Tabu search is a meta-heuristic approach which is associated with a method of seeking optimal solutions. But since most of the heuristic methods select moves in the neighborhood of the current solution improving the value of the objective function, this leads at times to get stuck at the local optima and terminate the procedure at the arrival of the local optimum instead of the global optimum.

The situation is shown in the figure below.

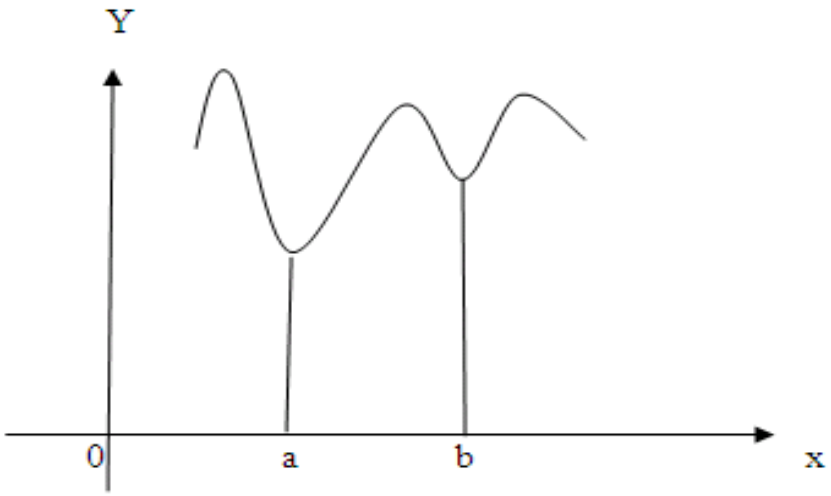


Figure 2.1 local heuristic search procedures and the solution space

Tabu search is used to get out of the local optimum. Hence, Tabu search guides local heuristic search procedures to explore the solution space beyond local optimality. The important characteristics of Tabu search are:

1. It allows moves, which even worsen of the objective function.
2. It prevents cycling temporarily,

   i.e., temporarily revisiting solutions examined recently by using Tabu list. Tabu list contains forbidden moves having been selected recently. The Tabu list changes as we move to the next solution whether it is better or worse.

3. It uses incumbent solution which is the best solution obtained so far.

The incumbent solution changes to better solution as and when we arrive at it.

4. The procedure terminates when we revisit an already examined solution.

2.3 Tabu Search Background and Relevance:

Faced with the challenge of solving hard optimization problems that abound in the real world, classical methods often encounter great difficulty. Vitally important applications in business, engineering, economics and science cannot be tackled with any reasonable hope of success, within practical time horizons, by solution methods that have been the predominant focus of academic research throughout the past three decades. The meta-heuristic approach called tabu search (TS) is dramatically changing our ability to solve problems of practical significance.

Current applications of TS span the realms of resource planning, telecommunications, VLSI design, financial analysis, scheduling, space planning, energy distribution, molecular engineering, logistics, pattern classification, flexible manufacturing, waste management, mineral exploration, biomedical analysis, environmental conservation and scores of others. In recent years, journals in a wide variety of fields have published tutorial articles and computational studies documenting successes by tabu search in extending the frontier of problems that can be handled effectively yielding solutions whose quality often significantly surpasses that obtained by methods previously applied.. A more comprehensive list, including summary descriptions of gains achieved from practical implementations, can be found in Glover and Laguna, 1997. [2]

2.4 General Tenets

The word *tabu* (or *taboo)* comes from Tongan, a language of Polynesia, where it was used by the Aborigines of Tonga island to indicate things that cannot be touched because they are sacred. According to Webster's Dictionary, the word now also means “a prohibition imposed by social custom as a protective measure" or of something "banned as constituting a risk.” These current more pragmatic senses of the word accord well with the theme of tabu search. The risk to be avoided in this case is that of following a counter-productive course, including one which may lead to entrapment

without hope of escape. On the other hand, as in the broader social context where “protective prohibitions” are capable of being superseded when the occasion demands, the “tabus” of tabu search are to be overruled when evidence of a preferred alternative becomes compelling.

The most important association with traditional usage, however, stems from the fact that tabus as normally conceived are transmitted by means of a social memory which is subject to modification over time. This creates the fundamental link to the meaning of "tabu" in tabu search. The forbidden elements of tabu search receive their status by reliance on an evolving memory, which allows this status to shift according to time and circumstance.

More particularly, tabu search is based on the premise that problem solving, in order to qualify as intelligent, must incorporate adaptive memory and responsive exploration. The adaptive memory feature of TS allows the implementation of procedures that are capable of searching the solution space economically and effectively. Since local choices are guided by information collected during the search, TS contrasts with memory less design that heavily rely on semi random processes that implement a form of sampling.

Examples of memory less methods include semi greedy heuristics and the prominent “genetic” and “annealing” approaches inspired by metaphors of physics and biology. Adaptive memory also contrasts with rigid memory designs typical of branch and bound strategies. (It can be argued that some types of evolutionary procedures that operate by combining solutions, such as genetic algorithms, embody a form of implicit memory. Special links with evolutionary methods, and implications for establishing more effective variants of them, are discussed in Glover and Laguna (1997).)

The emphasis on responsive exploration in tabu search, whether in a deterministic or probabilistic implementation, derives from the supposition that a bad strategic choice can yield more information than a good random choice. In a system that uses memory, a bad choice based on strategy can provide useful clues about how the strategy may profitably be changed. (Even in a space with significant randomness a purposeful design can be more adept at uncovering the imprint of structure.)

Responsive exploration integrates the basic principles of intelligent search, i.e., exploiting good solution features while exploring new promising regions. Tabu search is concerned with finding new and more effective ways of taking advantage of the mechanisms associated with both adaptive memory and responsive exploration. The development of new designs and strategic mixes makes TS a fertile area for research and empirical study. [2]

2.5 Use of Memory

The memory structures in tabu search operate by reference to four principal dimensions, consisting of recency, frequency, quality, and influence (Figure 2.2). Recency-based and frequency-based based memory complement each other, and have important characteristics we amplify in later sections. The quality dimension refers to the ability to differentiate the merit of solutions visited during the search. In this context, memory can be used to identify elements that are common to good solutions or to paths that lead to such solutions.

Operationally, quality becomes a foundation for incentive-based learning, where inducements are provided to reinforce actions that lead to good solutions and penalties are provided to discourage actions that lead to poor solutions. The flexibility of these memory structures allows the search to be guided in a multi-objective environment, where the goodness of a particular search direction may be determined by more than one function. The tabu search concept of quality is broader than the one implicitly used by standard optimization methods.

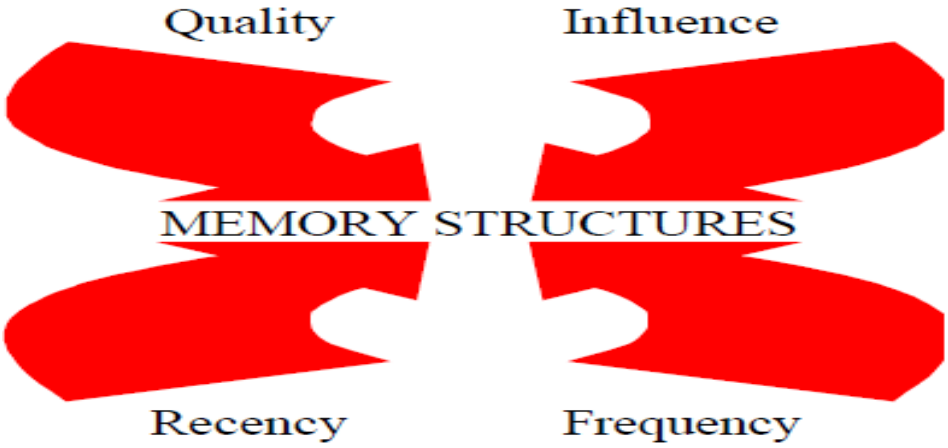


Figure 2.2 Four tabu search dimensions

The fourth dimension, influence, considers the impact of the choices made during the search, not only on quality but also on structure. (In a sense, quality may be regarded as a special form of influence.) Recording information about the influence of choices on particular solution elements incorporates an additional level of learning. By contrast, in branch and bound, for example, the separation rules are prespecified and the branching directions remain fixed, once selected, at a given node of a decision tree. It is clear however that certain decisions have more influence than others as a function of the neighborhood of moves employed and the way that this neighborhood is negotiated (e.g., choices near the root of a branch and bound tree are quite influential when using a depth-first strategy). The assessment and exploitation of influence by a memory more flexible than embodied in such tree searches is an important feature of the TS framework.

The memory used in tabu search is both explicit and attributive. Explicit memory records complete solutions, typically consisting of elite solutions visited during the search. An extension of this memory records highly attractive but unexplored neighbors of elite solutions. The memorized elite solutions (or their attractive neighbors) are used to expand the local search.

Alternatively, TS uses attributive memory for guiding purposes. This type of memory records information about solution attributes that change in moving from one solution to another. For example, in a graph or network setting, attributes can consist of nodes or arcs that are added, dropped or repositioned by the moving mechanism. In production scheduling, the index of jobs may be used as attributes to inhibit or encourage the method to follow certain search directions.

2.6 Intensification and Diversification

Two highly important components of tabu search are intensification and diversification strategies. Intensification strategies are based on modifying choice rules to encourage move combinations and solution features historically found good. They may also initiate a return to attractive regions to search them more thoroughly. Since elite solutions must be recorded in order to examine their immediate neighborhoods, explicit memory is closely related to the implementation of intensification strategies. As Figure 2.3 illustrates, the main difference between

intensification and diversification is that during an intensification stage the search focuses on examining neighbors of elite solutions.

Here the term “neighbors” has a broader meaning than in the usual context of “neighborhood search.” That is, in addition to considering solutions that are adjacent or close to elite solutions by means of standard move mechanisms, intensification strategies generate “neighbors” by either grafting together components of good solution or by using modified evaluation strategies that favor the introduction of such components into a current (evolving) solution.

Figure 2.3 Intensification and diversification

The diversification stage on the other hand encourages the search process to examine unvisited regions and to generate solutions that differ in various significant ways from those seen before. Again, such an approach can be based on generating subassemblies of solution components that are then “fleshed out” to produce full solutions, or can rely on modified evaluations as embodied, for example, in the use of penalty / incentive functions.

Intensification strategies require a means for identifying a set of elite solutions as basis for incorporating good attributes into newly created solutions. Membership in the elite set is often determined by setting a threshold which is connected to the objective function value of the best solution found during the search. However, considerations of clustering and “anti-clustering” are also relevant for generating such a set, and more particularly for generating subsets of solutions that may be used for specific phases of intensification and diversification. In the following sections, we show how the treatment of such concerns can be enhanced by making use of special memory structures. The TS notions of intensification and diversification are

beginning to find their way into other meta-heuristics. It is important to keep in mind that these ideas are somewhat different than the old control theory concepts of "exploitation" and "exploration," especially in their implications for developing effective problem solving strategies.

2.7 The Capacitated Plant Location Problem

The Capacitated Plant Location Problem (CPLP) is one of the basic problems in Location Theory. It is encountered in many application settings that involve locating facilities with limited capacity to provide services. The CPLP can be formally described as follows.

A set of customers I with demands d_i, $i \in I$, for some product are to be served from plants located in a subset of sites from a given set J of "potential sites". For each site $j \in J$, the fixed cost of "opening" the plant at j is f_j and its capacity is K_j. The cost of transporting one unit of the product from site j to customer i is c_{ij}. The objective is to minimize the total cost, i.e., the sum of the fixed costs for open plants and the transportation costs.

Letting x_{ij} ($i \in I$, $j \in J$) denote the quantity shipped from site j to customer i (the x_{ij}'s are the so-called *flow variables*) and y_j ($j \in J$) be a 0-1 variable indicating whether or not the plant at site j is open (the y_j's are the *location variables*), the problem can be formulated as the following mathematical program:

(CPLP) Minimize $z = \sum_{j \in J} f_j y_j + \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij}$

Subject to $\sum_{j \in J} x_{ij} = d_i, \; i \in I$

$$\sum_{i \in I} x_{ij} \le K_j y_j, \; j \in J$$

$$x_{ij} \ge 0, \; i \in I, \; j \in J$$

$$y_j \in \{0,1\}, \; j \in J$$

- For any vector $\tilde{\boldsymbol{y}}$ of location variables, optimal (w.r.t. to this plant configuration) values for the flow variables $\boldsymbol{x}(\tilde{\boldsymbol{y}})$ can be retrieved by solving the associated transportation problem:

(TP) Minimize $z(\tilde{\boldsymbol{y}}) = \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij}$

subject to $\sum_{j \in J} x_{ij} = d_i, \; i \in I$

$$\sum_{i \in I} x_{ij} \leq K_j \tilde{y}_j, \; j \in J$$

$$x_{ij} \geq 0, \; i \in I, \; j \in J$$

If $\tilde{\boldsymbol{y}} = \boldsymbol{y}^*$, the optimal location variable vector, the optimal solution to the original CPLP problem is simply given by $(\boldsymbol{y}^*, \boldsymbol{x}(\boldsymbol{y}^*))$.

- An optimal solution of the original CPLP problem can always be found at an extreme point of the polyhedron of feasible flow vectors defined by the constraints:

$$\sum_{j \in J} x_{ij} = d_i, \; i \in I$$

$$\sum_{i \in I} x_{ij} \leq K_j, \; j \in J$$

$$x_{ij} \geq 0, \; i \in I, \; j \in J$$

This property follows from the fact that the CPLP can be interpreted as a fixed-charge problem defined in the space of the flow variables. This fixed-charge problem has a concave objective function that always admits an extreme point minimum. The optimal values for the location variables can easily be obtained from the optimal flow vector by setting y_j equal to 1 if $\sum_{i \in I} x_{ij} > 0$, and to 0 otherwise.

To our knowledge, no TS heuristic has ever been proposed for the CPLP, but we will see in the following that this problem can be used to illustrate many important concepts related to the approach.

2.7.1 Search Space And Neighborhood Structure

As we just mentioned, TS is an extension of classical LS methods. In fact, basic TS can be seen as simply the combination of LS with short-term memories. It follows that the two first basic elements of any TS heuristic are the definition of its *search space* and its *neighborhood structure*.

The search space of an LS or TS heuristic is simply the space of all possible solutions that can be considered (visited) during the search. For instance, in the CVRP example described in Section 1, the search space could simply be the set of feasible solutions to the problem, where each point in the search space corresponds to a set of vehicles routes satisfying all the specified constraints. While in that case the definition of the search space seems quite natural, it is not always so. Consider now the CPLP example of Section 1:

The feasible space involves both integer location and continuous flow variables that are linked by strict conditions; moreover, as has been indicated before, for any feasible set of values for the location variables, one can fairly easily retrieve optimal values for the flow variables by solving the associated transportation problem. In this context, one could obviously use as a search space the full feasible space; this would involve manipulating both location and flow variables, which is not an easy task. A more attractive search space is the set of feasible vectors of location variables, i.e., feasible vectors in $\{0, 1\}^{|J|}$, any solution in that space being "completed" to yield a feasible solution to the original problem by computing the associated optimal flow variables.

It is interesting to note that these two possible definitions are not the only ones. Indeed, on the basis of Remark 2 above, one could also decide to search instead the set of extreme points of the set of feasible flow vectors, retrieving the associated location variables by simply noting that a plant must be open whenever some flow is allocated to it. In fact, this type of approach was used successfully by Crainic, Gendreau and Farvolden (2000) to solve the Fixed Charge Multi-commodity Network Design Problem, a more general problem that includes the CPLP as a special case. It is also important to note that it is not always a good idea to restrict the search space to

feasible solutions; in many cases, allowing the search to move to infeasible solutions is desirable, and sometimes necessary (see subsection 3.4 below for further details).

Closely linked to the definition of the search space is that of the neighborhood structure. At each iteration of LS or TS, the local transformations that can be applied to the current solution, denoted *S*, define a set of neighboring solutions in the search space, denoted *N(S)* (the neighborhood of *S*). Formally, *N(S)* is a subset of the search space defined by:

N(S) = {*solutions obtained by applying a single local transformation to S*}.

In general, for any specific problem at hand, there are many more possible (and even, attractive) neighborhood structures than search space definitions. This follows from the fact that there may be several plausible neighborhood structures for a given definition of the search space. This is easily illustrated on our CVRP example that has been the object of several TS implementations. In order to simplify the discussion, we suppose in the following that the search space is the feasible space.

Simple neighborhood structures for the CVRP involve moving at each iteration a single customer from its current route; the selected customer is inserted in the same route or in another route with sufficient residual capacity. An important feature of these neighborhood structures is the way in which insertions are performed: one could use random insertion or insertion at the best position in the target route; alternately, one could use more complex insertion schemes that involve a partial re-optimization of the target route, such as GENI insertions (see Gendreau, Hertz and Laporte, 1994). Before proceeding any further it is important to stress that while we say that these neighborhood structures involve moving a *single* customer, the neighborhoods they define contain all the feasible route configurations that can be obtained from the current solution by moving *any* customer and inserting it in the stated fashion. Examining the neighborhood can thus be fairly demanding.

More complex neighborhood structures for the CVRP, such as the λ-interchange of Osman (1993), are obtained by allowing simultaneously the movement of customers to different routes and the swapping of customers between routes. In Rego and Roucairol (1996), moves are defined by *ejection chains* that are sequences of coordinated movements of customers from one route to another; for instance, an

ejection chain of length 3 would involve moving a customer v_1 from route R_1 to route R_2, a customer v_2 from R_2 to route R_3 and a customer v_3 from R_3 to route R_4. Other neighborhood structures involve the swapping of sequences of several customers between routes, as in the Cross-exchange of Taillard *et al.* (1997). These types of neighborhoods have seldom be used for the CVRP, but are common in TS heuristics for its time-windows extension, where customers must be visited within a pre-specified time interval.

We refer the interested reader to Gendreau, Laporte and Potvin (2002) and Bräysy and Gendreau (2001) for a more detailed discussion of TS implementations for the CVRP and the Vehicle Routing Problem with Time-Windows.

When different definitions of the search space are considered for any given problem, neighborhood structures will inevitably differ to a considerable degree. This can be illustrated on our CPLP example. If the search space is defined with respect to the location variables, neighborhood structures will usually involve the so-called "*Add/Drop*" and "*Swap*" moves that respectively change the status of one site (i.e., either opening a closed facility or closing an open one) and move an open facility from one site to another (this move amounts to performing simultaneously an Add move and a Drop move). If, however, the search space is the set of extreme points of the set of feasible flow vectors, these moves become meaningless. One should instead consider moves defined by the application of pivots to the linear programming formulation of the transportation problem, since each pivot operation moves the current solution to an adjacent extreme point.

The preceding discussion should have clarified a major point: choosing a search space and a neighborhood structure is by far the most critical step in the design of any TS heuristic. It is at this step that one must make the best use of the understanding and knowledge he/she has of the problem at hand.

2.7.2 Tabus

Tabus are one of the distinctive elements of TS when compared to LS. As we already mentioned, tabus are used to prevent cycling when moving away from local optima through non-improving moves. The key realization here is that when this situation occurs, something needs to be done to prevent the search from tracing back its steps to where it came from. This is achieved by declaring *tabu* (disallowing) moves that

reverse the effect of recent moves. For instance, in the CVRP example, if customer v_1 has just been moved from route R_1 to route R_2, one could declare tabu moving back v_1 from R_2 to R_1 for some number of iterations (this number is called the *tabu tenure* of the move).

Tabus are also useful to help the search move away from previously visited portions of the search space and thus perform more extensive exploration.

Tabus are stored in a *short-term memory* of the search (the *tabu list*) and usually only a fixed and fairly limited quantity of information is recorded. In any given context, there are several possibilities regarding the specific information that is recorded. One could record complete solutions, but this requires a lot of storage and makes it expensive to check whether a potential move is tabu or not; it is therefore seldom used. The most commonly used tabus involve recording the last few transformations performed on the current solution and prohibiting reverse transformations (as in the example above); others are based on key characteristics of the solutions themselves or of the moves.

To better understand how tabus work, let us go back to our reference problems. In the CVRP, one could define tabus in several ways. To continue our example where customer v_1 has just been moved from route R_1 to route R_2, one could declare tabu specifically moving back v_1 from R_2 to R_1 and record this in the short-term memory as the triplet (v_1, R_2, R_1). Note that this type of tabu will not constrain the search much, but that cycling may occur if v_1 is then moved to another route R_3 and then from R_3 to R_1. A stronger tabu would involve prohibiting moving back v_1 to R_1 (without consideration for its current route) and be recorded as (v_1, R_1). An even stronger tabu would be to disallow moving v_1 to any other route and would simply be noted as (v_1).

In the CPLP, when searching the space of location variables, tabus on Add/Drop moves should prohibit changing the status of the affected location variable and can be recorded by noting its index; tabus for Swap moves are more complex: they could be declared with respect to the site where the facility was closed, to the site where the facility was opened, to both locations (i.e., changing the status of both location variables is tabu), or to the specific swapping operation.

When searching the space of flow variables, one can take advantage of the fact that a pivot operation is associated with a unique pair of entering and leaving variables to define tabus; while here again several combinations are possible, experience has

shown that when dealing with pivot neighborhood structures, tabus imposed on leaving variables (to prevent them from coming back in the basis) are usually much more effective.

Multiple tabu lists can be used simultaneously and are sometimes advisable. For instance, in the CPLP, if one uses a neighborhood structure that contains both Add/Drop and Swap moves, it might be a good idea to keep a separate tabu list for each type of moves.

It has been shown, however, that fixed-length tabus cannot always prevent cycling, and some authors have proposed varying the tabu list length during the search (Glover, 1989, 1990; Skorin-Kapov, 1990; Taillard, 1990 and 1991). Another solution is to randomly generate the tabu tenure of each move within some specified interval; using this approach requires a somewhat different scheme for recording tabus that are then usually stored as *tags* in an array (the entries in this array will usually record the iteration number until which a move is tabu; see Gendreau, Hertz and Laporte, 1994, for more details).

2.8 Aspiration Criteria

While central to TS, tabus are sometimes too powerful: they may prohibit attractive moves, even when there is no danger of cycling, or they may lead to an overall stagnation of the searching process. It is thus necessary to use algorithmic devices that will allow one to *revoke* (cancel) tabus. These are called *aspiration criteria*. The simplest and most commonly used aspiration criterion (found in almost all TS implementations) consists in allowing a move, even if it is tabu, if it results in a solution with an objective value better than that of the current best-known solution (since the new solution has obviously not been previously visited).

Much more complicated aspiration criteria have been proposed and successfully implemented (see, for instance, de Werra and Hertz, 1989, or Hertz and de Werra, 1991), but they are rarely used. The key rule in this respect is that if cycling cannot occur, tabus can be disregarded.

2.9 A Template For Simple Tabu Search

We are now in the position to give a general template for TS, integrating the elements we have seen so far. We suppose that we are trying to minimize a function *f(S)* over some domain and we apply the so-called "best improvement" version of TS, i.e., the version in which one chooses at each iteration the best available move (this is the most commonly used version of TS).

Notation

- ***S***, the current solution,
- ***S****, the best-known solution,
- ***f****, value of *S**,
- ***N(S)***, the neighborhood of *S*,
- ***Ñ(S)***, the "admissible" subset of *N(S)* (i.e., non-tabu or allowed by aspiration).

Initialization

Choose (construct) an initial solution S_0.

Set $S := S_0$, $f^* := f(S_0)$, $S^* := S_0$, $T := \emptyset$.

Search

While termination criterion not satisfied do

- Select *S* in *argmin* [*f(S')*]; $S' \varepsilon \tilde{N}(S)$
- if $f(S) < f^*$, then set $f^* := f(S)$, $S^* := S$;
- record tabu for the current move in *T* (delete oldest entry if necessary);

  end while.

2.10 Termination Criteria

One may have noticed that we have not specified in our template above a termination criterion. In theory, the search could go on forever, unless the optimal value of the problem at hand is known beforehand. In practice, obviously, the search has to be stopped at some point. The most commonly used stopping criteria in TS are:

- after a fixed number of iterations (or a fixed amount of CPU time);
- after some number of iterations without an improvement in the objective function value (the criterion used in most implementations);

- When the objective reaches a pre-specified threshold value.

In complex tabu schemes, the search is usually stopped after completing a sequence of *phases*, the duration of each phase being determined by one of the above criteria.

2.11 Probabilistic TS And Candidate Lists

In "regular" TS, one must evaluate the objective for every element of the neighborhood *N(S)* of the current solution. This can prove extremely expensive from the computational standpoint. An alternative is to instead consider only a random sample *N'(S)* of *N(S)*, thus reducing significantly the computational burden. Another attractive feature of this alternative is that the added randomness can act as an anti-cycling mechanism; this allows one to use shorter tabu lists than would be necessary if a full exploration of the neighborhood was performed. One the negative side, it must be noted that, in that case, one may miss excellent solutions. Probabilities may also be applied to activating tabu criteria.

Another way to control the number of moves examined is by means of *candidate list* strategies, which provide more strategic ways of generating a useful subset *N'(S)* of *N(S)*. (The probabilistic approach can be considered to be one instance of a candidate list strategy, and may also be used to modify such a strategy.) Failure to adequately address the issues involved in creating effective candidate lists is one of the more conspicuous shortcomings that differentiate a naive TS implementation from one that is more solidly grounded. Relevant designs for candidate list strategies are discussed in Glover and Laguna (1997).

CHAPTER 3
LITERATURE REVIEW

Over the last fifteen years, well over a hundred papers presenting applications of Tabu Search (TS), a heuristic method originally proposed by Glover in 1986, to various combinatorial problems have appeared in the operations research literature. In several cases, the methods described provide solutions very close to optimality and are among the most effective, if not the best, to tackle the difficult problems at hand. These successes have made TS extremely popular among those interested in finding good solutions to the large combinatorial problems encountered in many practical settings. Several papers, book chapters, special issues and books have surveyed the rich TS literature (a list of some of the most important references is provided in a later section). In spite of this abundant literature, there still seem to be many researchers who, while they are eager to apply TS to new problem settings, find it difficult to properly grasp the fundamental concepts of the method, its strengths and its limitations, and to come up with effective implementations. The purpose of this tabu search is to address this situation by providing an introduction in the form of a tutorial focusing on the fundamental concepts of TS. Throughout the material, two relatively straightforward, yet challenging and relevant, problems will be used to illustrate these concepts: the Classical Vehicle Routing Problem (CVRP) and the Capacitated Plant Location Problem (CPLP). These will be introduced in the following section. The remainder of the topic is organized as follows. The basic concepts of TS (search space, neighborhoods, and short-term tabu lists) are described and illustrated in Chapter 2.

3.1 What is Heuristic?

A heuristic algorithm is an algorithm that using a strategy that does not examine all possible solutions to a problem. Heuristic algorithms make no attempt to find the perfect solution to the problem. Instead, heuristic algorithms look for a "good enough" solution in an acceptable amount of time. A heuristic algorithm is one that will provide a solution close to the optimal, but may or may not be optimal. The concept

of heuristic solutions to problems normally solved via non-polynomial time algorithms has changed the way programmers regard NP and NP-Complete problems.

In computer science, a **Heuristic Algorithm** or simply a **Heuristic** is an algorithm that ignores whether the solution to the problem can be proven to be correct, but which usually produces a good solution or solves a simpler problem that contains or intersects with the solution of the more complex problem. Heuristics are typically used when there is no known way to find an optimal solution, or when it is desirable to give up finding the optimal solution for an improvement in run time.

Two fundamental goals in computer science are finding algorithms with provably good run times and with provably good or optimal solution quality. A **Heuristic** is an algorithm that abandons one or both of these goals; for example, it usually finds pretty good solutions, but there is no proof the solutions could not get arbitrarily bad; or it usually runs reasonably quickly, but there is no argument that this will always be the case.

For instance, say you are packing odd-shaped items into a box. Finding a perfect solution is a hard problem: there is essentially no way to do it without trying every possible way of packing them. What most people do, then, is "put the largest items in first, then fit the smaller items into the spaces left around them." This will not necessarily be perfect packing, but it will usually give packing that is pretty good. It is an example of a heuristic solution. The principal advantages of heuristic algorithms are that such algorithms are (often) *conceptually simpler* and (almost always) much *cheaper computationally* than optimal algorithms.

Example: **Assignment Problem**
Consider the following problem: 3 men are to be assigned to 3 jobs - where the assignment cost is given by the matrix below:

Job	**1**	**2**	**3**
Man **A**	1	3	4
B	3	7	4
C	3	4	2

Only one man can be assigned to one job and all the men should be assigned. What would be a heuristic algorithm for this problem?

We should stress here that a heuristic algorithm should be capable of being applied to the problem even if the costs in the above matrix are changed (i.e. a heuristic algorithm is a set of general rules for solving the problem that are independent of the particular data case being considered).

Heuristic for the Assignment Problem: One (simple) heuristic for the assignment problem would be: choose a man and a job at random. Assign the chosen man to the chosen job. Delete the chosen man and the chosen job from the problem and repeat with this new (smaller) problem.

This heuristic does not use any of the cost information and so we would not expect it to give very good results.

Note however the idea of *repetition*. This is a common concept in heuristic algorithms (both because it eases the task of programming the heuristic, and because if a certain algorithmic step is a good idea then why not repeats it?). A better heuristic might be: choose the smallest cost in the cost matrix (ties broken arbitrarily) and assign the corresponding man to the corresponding job - delete them from the problem and repeat with this new (smaller) problem. This heuristic would give the solution:
Cost 1 Assign A to job 1
Cost 2 Assign C to job 3
Cost 7 Assign B to job 2
Total cost 10

This illustrates a problem that often occurs with heuristics in that by the third assignment (of B to job 2) we have been "painted into a corner" by previous assignments and have little or no choice left (with the result that we have to assign B to job 2 at relatively high cost). Because of this problem a common idea with heuristics is the concept of *interchange* - the basic idea here is to juggle with the current solution to see if we can improve it e.g. with the solution above could we improve it by, for example, swapping the assignments of A and C thereby assigning A to job 3 and C to job 1? Here this swap is not worthwhile but some swaps (interchanges) are e.g. swap the assignments of A and B.

Note particularly here that we designed the above heuristic without ever having a mathematical formulation of the problem. It is difficult to imagine the variety of

existing computational tasks and the number of algorithms developed to solve them. Algorithms that either give nearly the right answer or provide a solution not for all instances of the problem are called heuristic algorithms. This group includes a plentiful spectrum of methods based on traditional techniques as well as specific ones.

3.2 Heuristics:

The word “heuristic” originated from the Greek root meaning to discover. In case of optimization of real life problems, the problems are encountered because of the highly complex nature. The regular algorithms are very often ineffective on these problems but there are, however, other approach that may be used to find a solution to models involving integer and or discrete variables. And, in many cases, such approaches have proven capable of providing acceptable solutions to truly massive size problems. Thus, the pragmatic or human approach can be considered to solve the problems and hence this shifts the whole paradigm from algorithm-based calculations to the employment of heuristic procedures or heuristic programming.

Heuristics are rules of thumb that are developed through intuition, experience and judgment. In artificial intelligence, a heuristic is a procedure that may lack a proof. It is used when the inter-relationships of the decision variables are not explicitly clear but there is some confidence in understanding the output for certain input. The result of application of heuristics cannot guarantee the optimal solution. A heuristic might help to find solutions which are good, but perhaps not the very best they can be. Obviously the measure of goodness and assessment of a heuristic technique is going to be relative to the domain, and to the specific job that problem solving is going to be applied in the domain. When one or more heuristics are combined with a procedure for deriving a solution from the associated rules, gives a heuristic program. Since the concept is to derive an acceptable solution and not the server-elusive optimal solution, thus the heuristics satisfy certain aspiration criteria as set by the decision maker.

The heuristic approaches are also being used to solve the real world bi-objective problems. These problems always attract us towards the goal programming techniques but of late another approach of the efficient optimal solutions has gained importance.

3.3 Different Heuristic Strategies

3.3.1 Exhaustive Search: The simplest of search algorithms is **exhaustive search** that tries all possible solutions from a predetermined set and subsequently picks the best one. Many problems have complexity worse than polynomial, such as the recursive Towers of Hanoi, which is ***O(2n)***. Some are inherently expensive, such as finding all permutations of a string of ***n*** characters, which is ***O(n!)***. Faced with an expensive algorithm, one should look for techniques to reduce the work done. We shall consider this with the example of a game-playing strategy that uses a search tree to look ahead some number of moves to determine the best move.

Example: A) Depth-First Search

- Always explore first child next
- Evaluate other children before root
- Use stack as data structure to hold pending nodes (or use recursive code)
- Goes deep into search tree quickly
- If tree has infinite levels, DFS may not terminate
- Aggressive attack, good if finite depth and multiple solutions

Order of Evaluation of Nodes in Search Tree

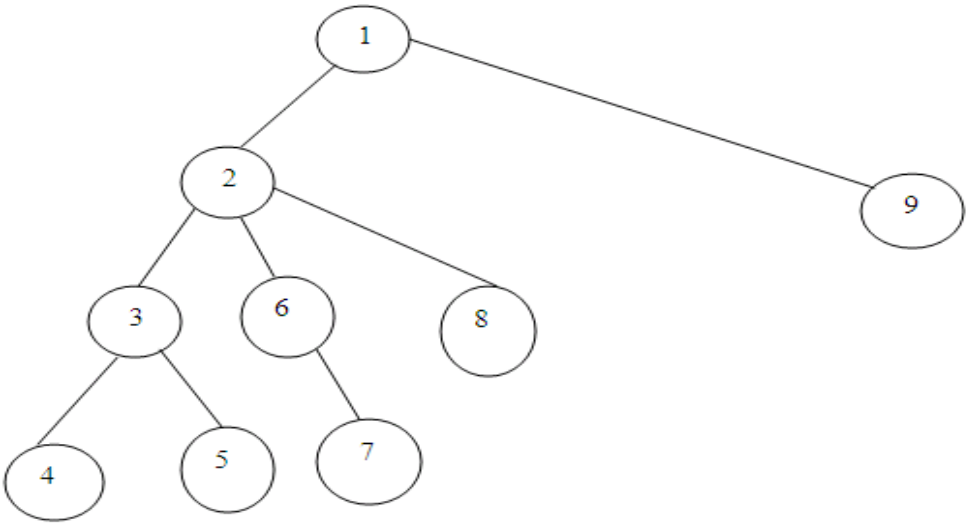


Figure 3.1: Shows order oh evaluation of nodes in DFS

B) Breadth-First Search

Explore all nodes at given level before descending to next

- Use queue as data structure
- BFS is slower but safer than DFS, especially if there is no guarantee on number of levels in search tree

Order of Evaluation of Nodes in Search Tree

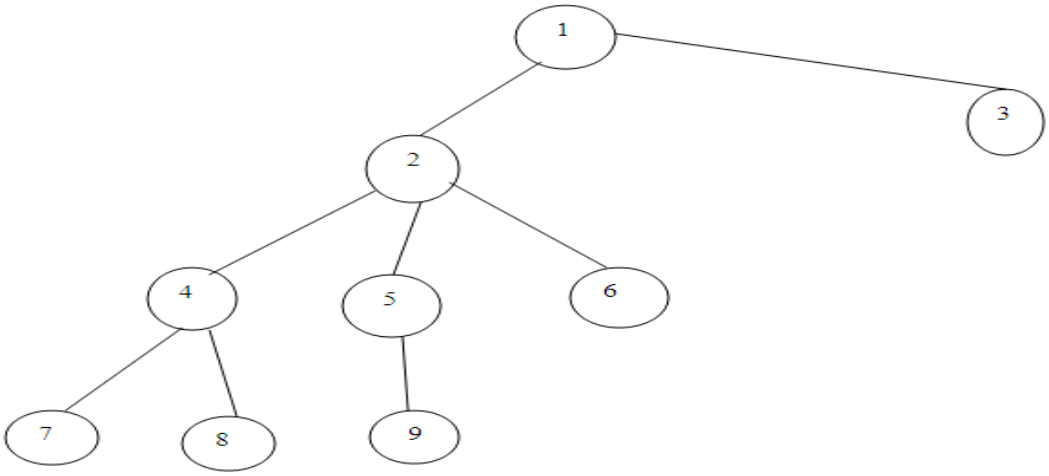


Figure 3.2: Shows order of evolution of nodes in BFS

3.3.2 Local Search

Local Search is a version of exhaustive search that only focuses on a limited area of the search space. Local search can be organized in different ways. Popular hill-climbing techniques belong to this class. Such algorithms consistently replace the current solution with the best of its neighbours if it is better than the current.

In **Hill Climbing** the basic idea is to always head towards a state which is better than the current one. So, if you are at town A and you can get to town B and town C (and your target is town D) then you should make a move if town B or C appear nearer to town D than town A does. In *steepest ascent* hill climbing you will always make your next state the best successor of your current state, and will only make a move if that successor is better than your current state.

This can be described as follows:

- Start with *current-state* = initial-state.
- Until *current-state* = goal-state OR there is no change in *current-state* do:

- Get the successors of the current state and use the evaluation function to assign a score to each successor.
- If one of the successors has a better score than the current-state then set the new current-state to be the successor with the best score.

Note that the algorithm does not attempt to exhaustively try every node and path, so no node list or agenda is maintained - just the current state. If there are loops in the search space then using hill climbing you shouldn't encounter them - you can't keep going up and still get back to where you were before. Hill climbing terminates when there are no successors of the current state, which are better than the current state itself.

3.3.3 Divide and Conquer

Divide and Conquer algorithms try to split a problem into smaller problems that are easier to solve. Solutions of the small problems must be combinable to a solution for the original one. This technique is effective but its use is limited because there is no a great number of problems that can be easily partitioned and combined in a such way. Examples:

- Binary search
- Powering a number
- Fibonacci numbers
- Matrix multiplication
- Stassen‘s algorithm, VLSI tree layout, etc……

3.3.4 Branch-and-Bound

Branch-and-Bound technique is a critical enumeration of the search space. It enumerates, but constantly tries to rule out parts of the search space that cannot contain the best solution.

Example: **4-Queens Problem**

FIFO branch-and-bound algorithm

- Initially, there is only one live node; no queen has been placed on the chessboard
- The only live node becomes E-node
- Expand and generate all its children; children being a queen in column 1, 2, 3, and 4 of row 1 (only live nodes left)
- Next E-node is the node with queen in row 1 and column 1

- Expand this node, and add the possible nodes to the queue of live nodes
- Bound the nodes that become dead nodes

Compare with backtracking algorithm

- Backtracking is superior method for this search problem

Where

Live node is a node that has been generated but whose children have not yet been generated.

E-node is a live node whose children are currently being explored. In other words, an E- node is a node currently being expanded.

Dead node is a generated node that is not to be expanded or explored any further.

All children of a dead node have already been expanded.

Branch-and-bound refers to all state space search methods in which all children of an E-node are generated before any other live node can become the E-node

3.3.5 Dynamic Programming

Dynamic Programming is an exhaustive search that avoids re-computation by storing the solutions of sub problems. The key point for using this technique is formulating the solution process as a recursion. Dynamic Programming is a recursive method for solving *sequential decision problems* (hereafter abbreviated as SDP). Also known as *backward induction*, it is used to find *optimal decision rules* in ―games against nature and *subgame perfect equilibria* of dynamic multi-agent games, and competitive equilibria in dynamic economic models. Dynamic programming has enabled economists to formulate and solve a huge variety of problems involving sequential decision making under uncertainty, and as a result it is now widely regarded as the single most important tool in economics.

Examples: a). Longest Common Subsequence (LCS)

Given two sequences x [1. . m]and y[1 . . n], find a longest subsequence common to them both.

b). Optimal Substructure

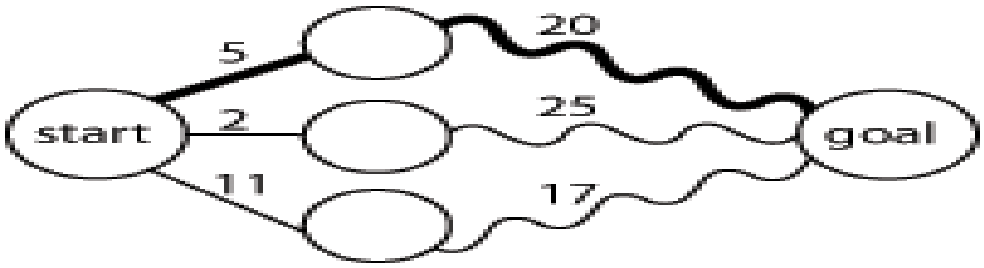


Figure 3.3

Figure3.3. Finding the shortest path in a graph using optimal substructure; a straight line indicates a single edge; a wavy line indicates a shortest path between the two vertices it connects (other nodes on these paths are not shown); the bold line is the overall shortest path from start to goal.

c). Overlapping Sub problems (Fibonacci sequence)

3.3.6 Greedy Technique

A popular method to construct successively space of solutions is *greedy technique*, that is based on the evident principle of taking the (local) best choice at each stage of the algorithm in order to find the global optimum of some objective function. A technique used in solving optimization problems. Typically, we are given a set of n inputs and the goal is to find a subset (or some output) that satisfies some constraints. Any subset (or output) that satisfies these constraints is called a feasible solution. In an optimization problem, we need to find a feasible solution that maximizes or minimizes a given objective function. A feasible solution that does this is called an optimal solution. The greedy technique works in stages, considering one input at a time (typically in some clever order). At each stage, a decision is made depending on whether it is best at this stage. For example, a simple criterion can be whether adding the current input will lead to an infeasible solution or not. Thus, a locally optimal choice is made in the hope that it will lead to a globally optimal solution.

Basic steps to finding efficient greedy algorithms:

- Start by finding a dynamic programming style solution
- Prove that at each step of the recursion, the min/max can be satisfied by a ”greedy choice”(*greedy substructure*)
- Show that only one recursive call needs to be made once the greedy choice is assumed. This is often natural when all the recursive calls are made by the min/max.
- Find the recursive solution using the greedy choice
- Convert to an iterative algorithm if possible

More generally, taking the direct approach:

- Show the problem is reduced to a sub problem via a greedy choice
- Prove there is an optimal solution containing the greedy choice
- Prove that combining the greedy choice with an optimal solution for the remaining sub problem yields an optimal solution

Example:

- For the MST problem: Prim‘s and Kruskal‘s algorithms
- For the SSSP problem: Dijkstra‘s algorithm

Remember, Dijkstra only works for graphs with no negative edge weights. Usually heuristic algorithms are used for problems that cannot be easily solved.

3.4 Heuristic Techniques

Branch-and-bound technique and **dynamic programming** are quite effective but their time-complexity often is too high and unacceptable for NP-complete tasks.

3.4.1 Hill-Climbing

Hill-climbing algorithm is effective, but it has a significant drawback called ***premature convergence***. Since it is ”greedy”, it always finds the nearest local optima of low quality. The goal of modern heuristics is to overcome this disadvantage.

Premature convergence: When a genetic algorithms population converges to something, which is not the solution, we wanted.

3.4.2 Simulated Annealing

Simulated Annealing Algorithm [20], invented in 1983, uses an approach similar to hill climbing, but occasionally accepts solutions that are worse than the current. The probability of such acceptance is decreasing with time.

Simulated Annealing (SA) is a generic probabilistic meta-algorithm for the global optimization problem, namely locating a good approximation to the global minimum of a given function in a large search space. It is often used when the search space is discrete (e.g., all tours that visit a given set of cities). For certain problems, simulated annealing may be more effective than exhaustive enumeration — provided that the goal is merely to find an acceptably good solution in a fixed amount of time, rather than the best possible solution.

The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one.

Some Real Applications of Simulated Annealing:

- Determining the sequence of observations for an automated astronomical telescope.
- Computer Aided Geometric Design.
- Optimization of Econometric Statistical Functions.
- Games with random moves determined by the simulated annealing algorithm.
- Arranging connections on chips and switching devices in telephone networks.

The SA Algorithm: In the Simulated Annealing algorithm, an objective function to be minimized is defined. Here it will be the total path length through a set of points. The distance between each pair of points is equivalent to the "energy" of a molecule. Then, "temperature" is the average of these lengths. Starting from an initial point, the algorithm swaps a pair of points and the total "energy" of the path is calculated.

3.4.3 Tabu Search

Tabu search [3] extends the idea to avoid local optima by using memory structures. The problem of simulated annealing is that after "jump"the algorithm can simply repeat its own track. Tabu search prohibits the repetition of moves that have been made recently. Webster's dictionary defines the word *tabu* or *taboo* to mean banned on grounds of morality or taste or as constituting a risk...". Tabu Search (TS) is an optimization method designed to help a search negotiate difficult regions (i.e. to escape from local minima or to cross-infeasible regions of the search space) by imposing restrictions. It was originally developed as a method for solving combinatorial optimization problems (these are problems where the control variables are some form of ordered list the *Travelling Salesman Problem* is the classic example), but variants to solve continuous and integer optimization problems have also been developed.

Example: ***The Classical Vehicle Routing Problem***:

Vehicle Routing Problems have very important applications in the area of distribution management. As a consequence, they have become some of the most studied problems in the combinatorial optimization literature and large number of papers and books deal with the numerous procedures that have been proposed to solve them. These include several TS implementations that currently rank among the most effective. The *Classical Vehicle Routing Problem* (CVRP) is the basic variant in that class of problems. It can formally be defined as follows. Let $G = (V, A)$ be a graph where V is the vertex set and A is the arc set. One of the vertices represents the *depot* at which a fleet of m identical vehicles of capacity Q is based, and the other vertices customers that need to be serviced. With each customer vertex v_i are associated a demand q_i and a service time t_i. With each arc (v_i, v_j) of A are associated a cost c_{ij} and a travel time t_{ij}. The CVRP consists in finding a set of routes such that:

- Each route begins and ends at the depot;
- Each customer is visited exactly once by exactly one route;
- The total demand of the customers assigned to each route does not exceed Q;
- The total duration of each route (including travel and service times) does not exceed a specified value L;
- The total cost of the routes is minimized.

A feasible solution for the problem thus consists in a partition of the customers into m groups, each of total demand no larger than Q, that are sequenced to yield routes (starting and ending at the depot) of duration no larger than L.

Scheduling ➢ Flow-Time Cell Manufacturing ➢ Heterogeneous Processor Scheduling ➢ Workforce Planning ➢ Classroom Scheduling ➢ Machine Scheduling ➢ Flow Shop Scheduling ➢ Job Shop Scheduling ➢ Sequencing and Batching **Design** ➢ Computer-Aided Design ➢ Fault Tolerant Networks ➢ Transport Network Design ➢ Architectural Space Planning ➢ Diagram Coherency ➢ Fixed Charge Network Design ➢ Irregular Cutting Problems **Location and Allocation** ➢ Multi commodity Location/Allocation ➢ Quadratic Assignment ➢ Quadratic Semi-Assignment ➢ Multilevel Generalized Assignment ➢ Lay-Out Planning ➢ Off-Shore Oil Exploration **Logic and Artificial Intelligence** ➢ Maximum Satisfiability ➢ Probabilistic Logic ➢ Clustering ➢ Pattern Recognition/Classification ➢ Data Integrity ➢ Neural Network |Training and Design **Technology** ➢ Seismic Inversion ➢ Electrical Power Distribution ➢ Engineering Structural Design ➢ Minimum Volume Ellipsoids ➢ Space Station Construction ➢ Circuit Cell Placement	**Telecommunications** ➢ Call Routing ➢ Bandwidth Packing ➢ Hub Facility Location ➢ Path Assignment ➢ Network Design for Services ➢ Customer Discount Planning ➢ Failure Immune Architecture ➢ Synchronous Optical Networks **Production, Inventory and Investment** ➢ Flexible Manufacturing ➢ Just-in-Time Production ➢ Capacitated MRP ➢ Part Selection ➢ Multi-item Inventory Planning ➢ Volume Discount Acquisition ➢ Fixed Mix Investment **Routing** ➢ Vehicle Routing ➢ Capacitated Routing ➢ Time Window Routing ➢ Multi-Mode Routing ➢ Mixed Fleet Routing ➢ Traveling Salesman ➢ Traveling Purchaser **Graph Optimization** ➢ Graph Partitioning ➢ Graph Coloring ➢ Clique Partitioning ➢ Maximum Clique Problems ➢ Maximum Planner Graphs ➢ P-Median Problems **General Combinational Optimization** ➢ Zero-One Programming ➢ Fixed Charge Optimization ➢ Non convex Nonlinear Programming ➢ All-or-None Networks ➢ Bi-level Programming ➢ General Mixed Integer Optimization

Table3.1: Applications of Tabu search algorithm

3.4.4 Swarm Intelligence

Swarm Intelligence was introduced in 1989. It is an artificial intelligence technique, based on the study of collective behavior in decentralized, self-organized, systems. Two of the most successful types of this approach are Ant Colony

Optimization (ACO) and Particle Swarm Optimization (PSO). In ACO artificial ants build solutions by moving on the problem graph and changing it in such a way that future ants can build better solutions. PSO deals with problems in which a best solution can be represented as a point or surface in an n-dimensional space. The main advantage of swarm intelligence [21] techniques is that they are impressively resistant to the local optima problem.

The typical swarm intelligence system has the following properties:

- It is composed of many individuals;
- The individuals are relatively homogeneous (i.e., they are either all identical or they belong to a few typologies);
- The interactions among the individuals are based on simple behavioral rules that exploit only local information that the individuals exchange directly or via the environment (stigmergy);
- The overall behavior of the system results from the interactions of individuals with each other and with their environment, that is, the group behavior self organizes.

The characterizing property of a swarm intelligence system is its ability to act in a coordinated way without the presence of a coordinator or of an external controller. Many examples can be observed in nature of swarms that perform some collective behavior without any individual controlling the group, or being aware of the overall group behavior. Notwithstanding the lack of individuals in charge of the group, the swarm as a whole can show an intelligent behavior. This is the result of the interaction of spatially neighboring individuals that act on the basis of simple rules.

Few examples of scientific and engineering swarm intelligence studies are given below:

- Clustering Behavior of Ants
- Nest Building Behavior of Wasps and Termites
- Flocking and Schooling in Birds and Fish
- Ant Colony Optimization
- Particle Swarm Optimization
- Swarm-based Network Management
- Cooperative Behavior in Swarms of Robots

In mathematical programming, considerable attention has been devoted to the un-capacitated facility location problem (UFLP). Krarup and Pruzan(1983) presented a simple plant location problem. Efroymson and Ray(1966) proposed a formulation of the un-capacitated plant location problem and the use of branch and bound algorithm to solve it. Khumawala(1972) utilized the special structure of UFLP to improve the branch and bound algorithm of Efroymson and Ray (1966). Bilde and Krarup(1977) and Erlenkotter(1978) developed a dual-based branch and bound procedure for the problem and this procedure has been widely accepted as an efficient known procedure. So, one of the main results has been the development of linear programming-based branch and bound algorithms. The standard reference in this area is the algorithm of Erlenkotter(1978), a branch and bound algorithm based on dual descent which is an efficient way to solve the un-capacitated warehouse location problem.

Guignard(1977) proposed to strengthen the separable Legrangian relaxation of UFLP by using Benders inequalities generated during a Lagrangian dual ascent procedure. Galvao and Raggi(1989) proposed a method for solving to optimally un-capacitated location problem. Korkel(1989) showed the exact solution of large-scale simple plant location problem to modify primal-dual version of Erlenkotter's(1978) exact algorithm. Approximation algorithms for the un-capacitated warehouse location have also been studied heavily. Beasely(1993) proposed Legrangian heuristics for location problems. Simao and Thizy(1989) developed a dual simplex algorithm for the canonical representation of UFLP.

Conn and Cornuejols(1990) proposed a projection also exploiting a dual formulation. Gao and Robinson(1994) presented a general model and dual-based and branch and bound solution procedure to find optimal solutions for several un-capacitated location problems that include UFLP.

They claimed that their proposed solution procedure effectively solves realistically sized UFLP. Al-sultan and Al-Fawzan(1999) presented a Tabu search approach to the UFLP. The best neighbor which is not Tabu is selected as the next state if it improves the objective function.

The main step is executed for a number of iterations. The algorithm also uses a sophisticated and effective heuristic as the starting point of the Tabu search.

Michel and Hentenryck(2004) presented an simple, yet robust and efficient, Tabu-search algorithm for the un-capacitated warehouse location problem(UWLP). The algorithm finds optimal solutions very quickly and with high frequencies. It also compares favorably with state-of-art genetic algorithms and should be a very valuable addition to the repertoire of tools for un-capacitated warehouse location due to its simplicity and effectiveness.

Ignizio and Cavalier(1994) have considered the problem of selecting upto a fixed number of sites from among a given number of potential sites for locating warehouses at them and clustering customers to the selected sites in such a way that each customer is assigned to a unique selected site. The single objective of this problem was to minimize the sum of distances from each customer to his/her assigned site. Praveena et al(1999) have extended this problem which selects upto a fixed number of sites from among a given number of potential sites for warehouse for clustering ration shops to them subject to several constraints with two objectives.

The two objectives are to minimize the total cost and duration of meeting the requirements of all the ration shops from their assigned warehouses at the selected sites. These two objectives are not accorded priorities. One constant is that each ration shop should be clustered to is that the set up cost of the warehouses should not exceed a certain budgetary amount. This problem has been solved by finding the set of efficient solutions of it using heuristic method consisting of a combination of add and drop rules.

Prakash et al. (2009) has developed a heuristic iterative algorithm incorporating Tabu search to find the set of efficient solutions of this problem.

The capacitated facility location problem (CFLP) has been studied extensively. Many exact algorithms and heuristic methods have been developed to solve it in last 40 years. Because UFLP and CFLP are closely related, many heuristic methods developed for the UFLP are also extended to the CFLP.

Kuehn and Hamburger (1963) developed the first heuristic method for the UFLP, which was later extended to the CFLP by Jacobsen (1983). This heuristic method consists of two phases, the first phase, called ADD, starts with all facilities closed and then the facility that causes the maximum total cost reduction is opened and this

phase ends when no more facilities can be opened to further reduce the total cost. The second phase is a local search procedure in which an open facility and a closed facility exchange their status if this exchange reduces the total-cost. Domschke and Drex (1985) proposed priority rules for the ADD procedure to improve its performance in cases where the facilities have distinct capacities and/or distinct fixed operating costs. Feldman et al. (1996) proposed a different strategy for the first phase, named DROP that was also extended to the CFLP by Jacobsen(1983).

Lagrangian relaxation has also been applied to several facility location problems. Cornuejols et al. (1991) presented an excellent analysis of all possible Lagrangian relaxations and the linear programming relaxation for the CFLP. Barahona and Chudak(2001) proposed a Lagrangian heuristic method for the UFLP and CFLP.

Several exact algorithms based on branch and bound have been proposed. The major differences among these algorithms are in the types of relaxation, the methods of solving the relaxed problem and the strategies to improve the lower bound. Leung and Magnanti (1989) introduced a family of faults and valid inequalities for solving the CFLP with equal capacities. Aardal(1998) proposed new valid inequalities and implemented two branch-and cut algorithms that are tested on small and medium test problems from the literature.

The capacitated warehouse problem consists of the well known transportation problem with the additional feature of affixed charge associated with each warehouse which is put to use. The problem is usually solved as a special type of mixed integer program me, so that relaxation and lower bounding are a vital part of any algorithm. A deeper insight into the relaxation process may eventually lead to more efficient algorithms for the problem. Baker (1982) has shown that the LP taxation of the capacitated warehouse location problem can incorporate constraints of a much more general nature than those previously described.

Kelly and Khumawala (1982) presented an algorithm for finding a minimal cost warehouse system design wherein individual warehouse have limited capacities and exhibit economies of scale because earlier the mixed integer-linear models that have been used in most analysis of warehouse location problems fails to capture the potential operating efficiencies associated with large scale facilities. The iterative

procedure defines and solves a series of conventional transportation problems in order to converge on the optimal system design.

Sun (2008) developed a Tabu search heuristic procedure for the capacitated facility location problem is developed, implemented and computationally tested. A specialized transportation algorithm is developed and employed to exploit the problem structure in solving transportation problems. The performance of the heuristic procedure is tested through computational experiments using test problems from the literature and new test problems from the literature and new test problems randomly generated. It found optimal solutions for almost all test problems used. As compared to the Lagrangean and the surrogate/Lagrangean heuristic methods, the Tabu search heuristic procedure found much better solutions.

Prakash and Aggarwal(1992) considered the problem of establishing a route between two specified nodes through a network with two objectives-one primary and another secondary is considered. The primary objective is to minimize the total cost of travel and the secondary objective is to minimize the duration of travel.

Prakash and Om(1996) has developed an algorithm to obtain the set of non-dominated solutions of the two-objective problem of determined programmes for augmentation of capacitates of depots and shipment of buses from them to the starting points of routes along with determining the capacity in reserve and the number of buses to be parked at the respective depots after the augmentation of their capacities is considered with two objectives.

One objective is to minimize the present value of the total cost of dead-travelling to be performed by buses between the depots and the starting points of their routes over a planning horizon plus the capital expenditure to be incurred in augmenting capacities of the depots. The other objective is to minimize the maximum distance among the distances traversed by individual buses from depots to the starting points of their respective routes. The two objectives are not accorded priorities.

A detailed discussion about efficient solution can be found in works of Ignizio (1982) and Steuer(1986). Some algorithms are described for solving plant location problems with lnon-linear warehousing costs, The heuristic problems are flexible with respect to the type of warehousing cost structure permitted, and may be used to

solve fixed p-median location problems as well as problems in which the numbers and locations of warehouses in solution are jointly determined as a trade-off between transportation and fixed and operating plant costs. Computational experience is reported by R.A.Whitaker (1985) on some well known problem sets in which the economies of scale in production are continuously concave. Comparisons with other solution methods indicate that the proposed procedures perform as well as or better than any currently known on these standard test problems.

Kratica et.al(2001) is of special interest as it contains a comprehensive and excellent description of a genetic algorithm and its comparison to Erlenkotter(1978). In many distribution systems, the location of the distribution facilities and the routing of the vehicles from these facilities are interdependent. The location routing problem(LRP), which combines the facility location and the vehicle routing decisions, is NP-hard. Due to the problem complexity, simultaneous solutions methods are limited to heuristics.

Tuzun and Burke(1999) presented two-phase Tabu search architecture for the solution of the LRP. First introduced in this paper, the two-phase approach offers a compatutationally efficient strategy that integrates facility location and routing decisions.

This tow-phase architecture makes it possible to search the solution space efficiently, thus producing good solutions without excessive computation. An extensive computational study shows that the Tabu search algorithm achieves significant improvement over a recent effective LRP heuristic.

The Tabu search meta-heuristic has been successfully applied to variety of combinational optimization problems, but not much research has been reported in using this meta-heuristic to solve the CFLP. The Tabu search heuristic procedure proposed by Grolimund and Ganascia(1997) was applied to the CFLP and limited computational results were reported. However, Tabu search procedure have been developed for more complicated facility location problems by many investigators such as Ferreira Filho and Galvao1998), Franca et.al(1999), Ghosh(2003) and Sun(2006a).

In 2010 using un-capacitated facility location problem to calculate the robust fault tolarence to minimize the sum of the cost of opening the facilities in R(robust) and the cost of assigning all node demands to open facilities. shiri chechik and david peleg.[4]

4.1Problem Statement:

The un-capacitated facility location problem (UFLP) can be stated as follows:
Given *n* possible sites and demands at *m* locations, determine the optimal location of facilities to fulfill all demands such that the total cost of establishing the facilities and fulfilling the demands (distribution cost) is minimized and maximized duration of establishing facilities vice versa.

4.1.1 Formulation Of Problem:

Suppose that there are *N* possible sites and demands at *M* locations, every site can be selected from among the *N* potential sites for locating customers at them. The *N* sites are to be clustered up to one or more sites in such a way that each shop is assigned to a unique site which is selected for locating a customer. There is no restriction on the number of site to be clustered to a selected customer. Let c_{ij} and t_{ij} $(i = 1,2,\ldots,M;\ j = 1,2,\ldots,N)$ units be the cost and time respectively of meeting requirements of site i from customer j. Let c_j units be the setup cost of a customer at the customer j only. Let x_{ij} be the variable assuming value 0 or 1 according as site i is not assigned or assigned to customer j, and y_j be the variable assuming the value 0 or 1 according as customer j is not selected or selected for locating a customer at it . Let C and T denote the total cost and duration respectively of meeting requirements of all the sites from their assigned customers. The mathematical formulation of this problem is as follows. The two objective functions which are sought to be minimized as well as maximized.

Minimized $\sum_{i=1}^{n}\sum_{j=1}^{m} C_{ij} + \sum_{i=1}^{n} f_i y_i$ .....(1)

$$T = \max\{t_{ij}\, x_{ij} : i = 1,2,\ldots,M;\ j = 1,2,\ldots,N\} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(2)$$

Constraints of the problem are

$$\sum_{i=1}^{n} x_{ij}=1, j=1,\ldots\ldots\ldots m \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3)$$

$$x_{ij} \leq y_i \quad i=1,\ldots\ldots\ldots n, \; j=1\ldots\ldots\ldots m \quad \ldots\ldots\ldots\ldots\ldots\ldots (4)$$

$$x_{ij}, y_i = 0 \text{ or } 1, \; i=1,\ldots n, \; j=1,\ldots m \quad \ldots\ldots\ldots\ldots\ldots\ldots (5)$$

Where

C_{ij} = the cost meeting customer j's demand from facility i ;

f_i = the cost is established a facility at site i ;

$$x_{ij} = \begin{cases} 1 if \; customer \; j \; is \; served \; from \; site \; i, \\ 0 \qquad\qquad otherwise; \end{cases}$$

$$y_i = \begin{cases} 1 \; if \; a \; facility \; is \; established \; at \; i, \\ 0 \qquad\qquad otherwise \; ; \end{cases}$$

Note that the objective functions *C* provided by (1) and *T* provided by (2) are not accorded priorities. The constraint (3) ensures that one site can be selected from among the *N* possible sites while constraints (4) and (5) ensure that each site is assigned to a unique site which is selected for locating a customer.

Note that x_{ij} is a binary variable because it is assumed that the demand of customer *j*, *j* = 1, 2,…,*m*, is fulfilled by one and only one facility, say *k* (i.e. no partial fulfillment of demand is allowed), in which case $x_{kj} = 1$, $x_{ij} = 0$, for all $i = 1, 2,\ldots, n, i \neq k$. y_i is also a binary variable since a facility *i* is either established ($y_i = 1$), in which case the fixed charge f_i is incurred, or it is not established ($y_i = 0$) and in which case no fixed charge is incurred.

The UFLP can mathematically be decomposed into two interdependent subproblems:

1. Location. The facilities to be established are determined here (*i.e.*, y_i's).
2. Allocation. For those established facilities, determines the distribution pattern (*i.e.*, x_{ij}'s).

For any solution of the location sub problem, an optimal solution of the allocation sub problem is easily obtained. More specifically, for any given y vector, an optimal assignment of the x_{ij}'s can obtained by using the following formula:

k : min C_{kj} , k= 1, ...n.

$$x_{ij} = \begin{cases} 1 & if \; i = k \\ 0 & otherwise \end{cases}$$

Therefore, if y_i's are known, the x_{ij}'s can be optimally determined using the above argument. Krarup and pruzan[24] presented using a similer argument. We propose using the tabu search approach to find the optimal set of y_i's .

5. 1The Proposed Algorithm

As explained in chapter4, This thesis concentrate on the location sub problem since it is the hard part of the problem to solve. Our proposed algorithm uses the tabu search technique to generate a set of y's and for every y, we use the argument in chapter 4 to get the optimal allocation (i.e., x_{ij}'s). The algorithm can be summarized as follows: use any heuristic to get an initial solution (i.e., y) (one can also use a random starting solution, or use the net benefit heuristic proposed below). Compute its total cost, and store it as the best solution so far. Using this solution, generate a set of neighboring y's. For every y, find the optimal allocation (i.e. x_{ij}'s) by the argument in chapter 4, and find the resulting total cost. Then the best improving y (or best non-Tabu if no improving y is found) is selected and its associated attribute is stored in the Tabu list. This procedure is repeated and controlled by tabu search until a predetermined number of iterations have been performed.

5.1.1 The Net Benefit Heuristic *(NBH)*

To start the Tabu search algorithm, one needs to have an initial starting solution. It is preferred that this initial solution is of good quality and requires little computation time.

The NBH consists of two phases: the initial phase and the refinement phase. The *initial phase* proceeds as follows: for every demand j, j = 1, 2,…,m, find the facility that can supply this demand at the cheapest transportation cost (i.e. find i^* such that $c_{i^*j} = \min_{1 \le i \le n} c_{ij}$, and establish this facility if it has not already been established). The result of the initial phase is the establishment of a certain number of facilities that will satisfy all demands (this of course constitutes an upper bound on the optimal solution).

The refinement phase is a procedure in which each facility that was established in the initial phase is investigated for possible closure, whereby a facility is closed if this can result in a net saving. This happens when the saving due to the closure of the facility (i.e. the fixed charge) is higher than the extra distribution cost incurred by allocating the demand originally allocated to this facility to the next cheapest established facility.

The NBH as shown in section 6.4 is very good in generating a good initial solution (i.e. The optimal on average) using a quite small running time. The NBH is stated below.

5.1.2 The NBH Algorithm

- ***Initial Phase***

Step 1: For each demand j, j = 1, 2,…, m, find the facility that can supply this demand at the cheapest distribution cost and denote it by i *.

For all *j*, *j* = 1,…, *m*,

let $C_{i^*j} = \min_{1 \le i \le n} C_{ij}$, and form the set *V* of ordered pairs, $V = \{(i^*, j)\}$.

Step 2: Form the set I from V by extracting the first index. This is the list of facilities Suggested for establishment.

Let $I = \{i^* : (i^*, j) \in V\}$.

Step 3: Evaluate the current solution which constitutes an upper bound one (UB1) on the optimal solution.

Calculate UB 1 = $\sum_{(i,j)\in V} C_{ij} x_{ij} + \sum_{i \in I} f_i y_i$

Step4: Evaluate the current solution which constitutes an upper bound two (UB2) on the optimal solution

Calculate UB2 = $\sum_{(i,j)\in V} t_{ij} x_{ij}$

- ***Refinement Phase***

Step 1: Set *k* = 1, let *m** = $|I|$ or *m** is the number of established facilities at the initial phase.

Step 2: Consider the kth facility established in the initial phase.

Let *i* be the *k*th element in *I*.

Step 3: Let the set J be the indices of all those demands currently satisfied by facility i.

Let $J = \{ j \mid (i, j) \in V\}$.

Step 4: Repeat for all facilities established step 5, 6, 7

Step 5: For each demand currently satisfied by facility i, find which established facility other than i can satisfy this demand at minimum distribution cost and compute the extra cost due to this reallocation.

For each $j \in J$, calculate $d_j = C_{lj} - C_{ij}$, where $C_{lj} = \min C_{tj}$, $t \in I$ and $t \neq i$.

Step 6 : Compute the extra cost which is equal to the cost of reallocation of demands currently satisfied by facility i, less the saving due to the closure of facility i.

Calculate $\delta = \sum_{j \in J} d_j - f_i$ create the array and calculate the maximum in the array.

Max = $|\delta[]|$

Step 7: If the extra cost computed in step 5 is negative (i.e. benefit), then it is better to close the facility (step 8), otherwise, consider the next facility (step 9).

If $\delta < 0$, go to step 8; otherwise go to step 9.

Step 8(a): $I = I - \{i\}$. UB1 = UB1 – $|\delta|$. If $|I| = 1$, stop; otherwise go to step 1.

Step 8(b): $I = I - \{i\}$. UB2 = UB2 – $|\delta|$. If $|I| = 1$, stop; otherwise go to step 1.

Step 9: If $k = m^*$, stop; otherwise, set $k = k + 1$ and go to step 2.

Next, we formally present the proposed algorithm.

5.1.3 Statement Of The Un-capacitated Facility Location Tabu Search Algorithm (UFLTSA)

- **Initialization Step**

Choose *nbhsize* (the size of the neighborhood). Choose a suitable size for the *tabu list*, *TL*.
Choose *ITERMAX* (the maximum number of non improving iterations) (see section 5.3 for a description of these parameters as well as suitable values for them).
Use the NBH algorithm to get an initial *y*. Let $y_{current} = y$, $y_{min} = y$, $y_{best} = y$.
Let *totcost*(*y*) be the total cost of *y*.
Let $TL = \emptyset$, $BV = totcost(y)$.
Let $k = 1$ and go to the main step.

- **Main Step**

Step 1 . Generate nbhsize random solutions from $y_{current}$. Each solution is evaluated and the solution with the minimum total cost among the generated solutions is selected as y_{min}.
$min = \infty$.
For $h = 1$ to *nbhsize*, do
Obtain *y* from $y_{current}$ (see section 5.1).
For every *j*, $j = 1,\ldots, m$, find $k : \min_k C_{kj}$, $k = 1,\ldots, n$.
Let $x_{kj} = 1$, $x_{ij} = 0$, for $i = 1,\ldots, n$, $i \neq k$.
Evaluate *totcost*(*y*).
If *totcost*(*y*) < *min*, then *min* = *totcost*(*y*), $y_{min} = y$.

Step 2 . Check tabu status.

Check whether or not the solution ymin found in step 1 is in the tabu list.

2.1. l = 1.

2.2 If ($y_{min} \notin TL$) or ($y_{min} \in TL$ and $min < BV$), then go to step 3; otherwise,

replace l by $l + 1$. Let the lth best solution be y_{min} (i.e. this is the best solution among all generated solutions in this iteration excluding those considered earlier in this step) and repeat this step.

Step 3: Update current solution.

Replace the current solution by the new solution ymin, and the best objective function (BV) by min. Store ymin in the tabu list.

Let *ycurrent = ymin*.

Store *ymin* in *TL* (see section 5.2).

If $min \geq BV$, then go to step 4; otherwise, $BV = min$, $k = 0$, and go to step 4.

Step 4: Check stopping criterion.

If the stopping criterion is satisfied, stop; otherwise, perform another iteration.

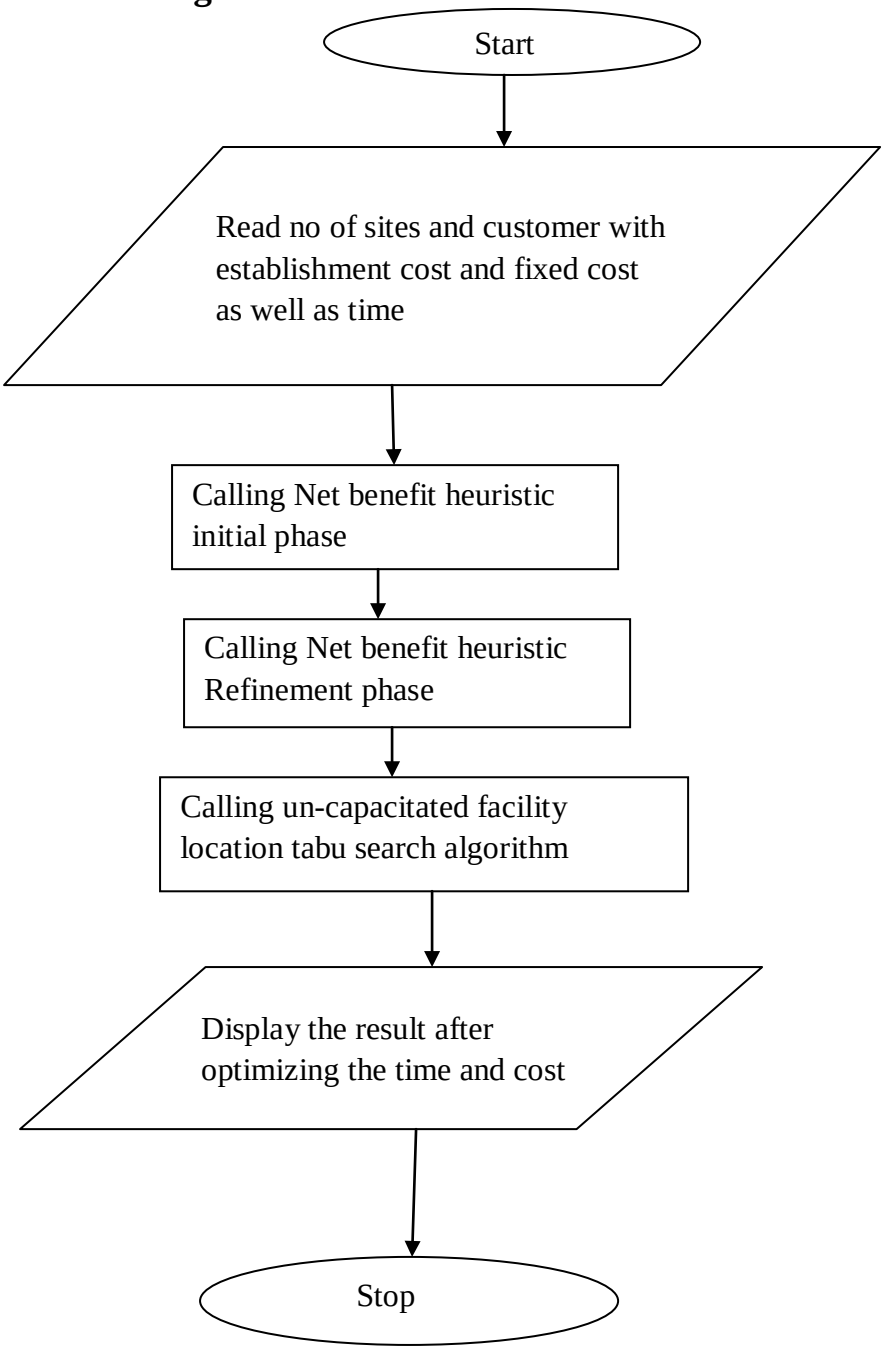
If k = ITERMAX, go to step 5; otherwise, replace k by k + 1, and go tostep1.

Step 5: Stop, and report the results.

Stop. y_{best} is the best solution found, and the corresponding total cost is BV.

Step 6: find t for y_{best} , x_{best}

5.3 Sequence Flow Diagram:



6. Computational Results And Discussion

In this section, This chapter discussed various implementations details and results of our computational experiments.

6.1. Generation Of Neighbors

In our experiments, we used the following scheme for generating a neighbor $y^{\wedge}$ for a given location vector y.

Let y_i be the ith component of the given vector y, where $1 \le i \le n$ and n is the number of facilities. To generate a neighbor y^, perform the following steps:

Step 1 . Let a be a given constant, where 0 < a < 1. A is considered a parameter of this scheme, and will be called *the probability threshold*.

Step 2. Set *k* = 1.

Step 3 .Generate a random number, b Î*U*(0, 1), where *U*(0, 1) is the uniform distribution between 0 and 1.

Step 4 . If b > a, go to step 5; otherwise, go to step 6.

Step 5. If *YK* = 1, set *y*^ *k* = 0, and if *yk* = 0, set *y*^ *k* = 1; stop.

Step 6 . If *k* = *n*, stop; otherwise set *k* = *k* + 1 and go to step 3.

6.2 Storing In The Tabu List

In our algorithm, the chosen neighbor (i.e. the move) has to be stored in the tabu list, *TL*, in step 3. This can be achieved by storing the value of the index *k* at the termination of the above scheme for generation of neighbors.

6.3. Parameter Setting

Tabu search is a parameter-sensitive technique similar to simulated annealing and genetic algorithms. As can be seen from the statement of the algorithm, it has the following parameters:

• The number of random solutions to be generated from the current one, *nbhsize*. Clearly, the larger the value of *nbhsize*, the better the quality of the solution, but the

more computation time is needed. Hence, one has to balance these two conflicting objectives.

• The tabu list size, *TL*. If *TL* is large, the many moves are made tabu and hence their reversal is forbidden, which makes the search emphasize diversification. When the tabu list size *TL* is small, moves that are made tabu at a certain iteration are allowed to be reversed after few iterations, which makes the search emphasize intensification.

• The probability threshold, α. The value of α affects the probability assigned to every facility to change its status. The higher the value of α, the more even is the chance given to facilities to change their status.

• Maximum number of non improving iterations, *ITERMAX*. This is the maximum number of consecutive non improving iterations that the algorithm is allowed to go through before termination. Of course, the higher the value of *ITERMAX*, the better the quality of the solution one expects, but of course at the expense of more computation time. We have conducted an extensive parametric study on the proposed algorithm. We started this study by testing the performance of the algorithm at different values of the four parameters nbhsize, α, *TL*, and *ITERMAX*. Clearly, as one increases the values of nbhsize, α and *ITERMAX*, the quality of the solution is expected to improve but, of course, at the expense of more computation time. Therefore, one has to strike a balance between these conflicting objectives.

The results of this study show that the following set of values give the best compromise between n the quality of the solution and the computation time. The higher range values are needed for larger problems. They definitely work for all other problems, but they take a little extra time compared to using lower range values.

nbhsize = 5*n*,

$\alpha \in [0.9,0.95]$,

tabu list size, $TL \in [7,15]$,

$ITERMAX \in [10,50]$.

6.4. Results And Discussion

Tables 1–3 show the results of our experiments. We coded the proposed algorithm in C++ and used a DELL personal computer equipped with Intel core”2 Duo processor with a speed of 2GHzThe following notation is used in tables 1–3.

Data file = the name of the data file as in the OR Library (Mahesh k. Sharma [13]).

n = number of facilities.

m = number of potential locations.

NBH% = the percentage deviation of the NBH solution from optimal.

NBH-T = the CPU time needed by NBH.

UFLTSA% = the percentage deviation of the UFLTSA solution from optimal.

UFLTSA-T = the CPU time needed by UFLTSA.

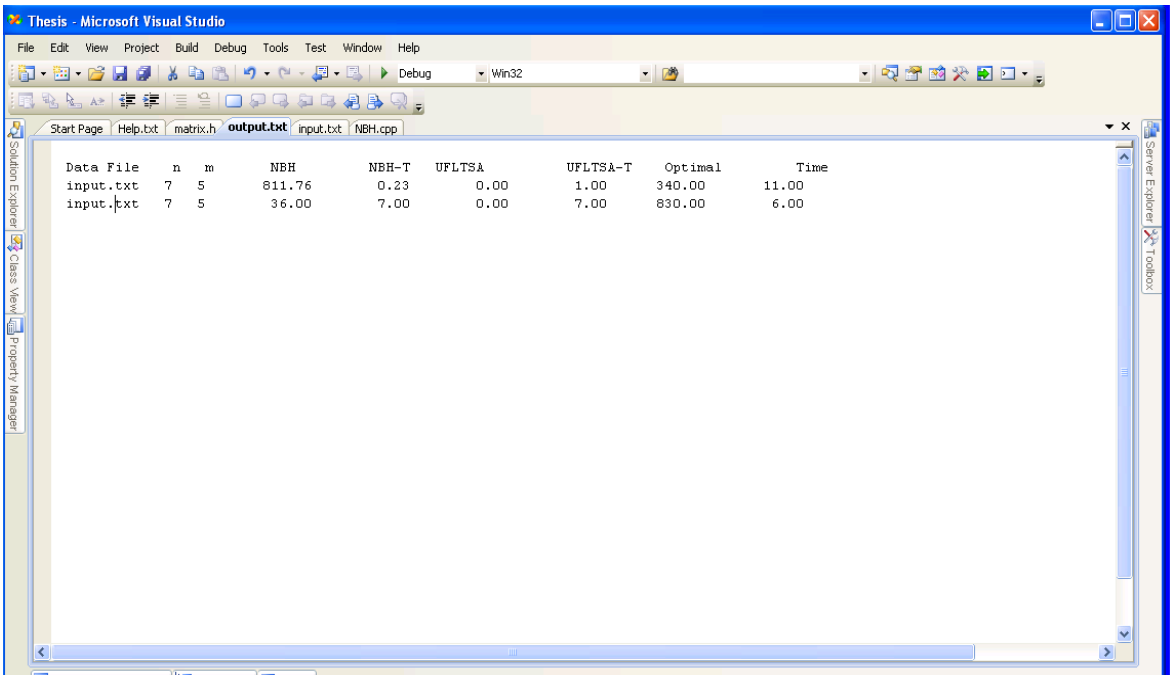


Table 6.1: **For a single site establishment**

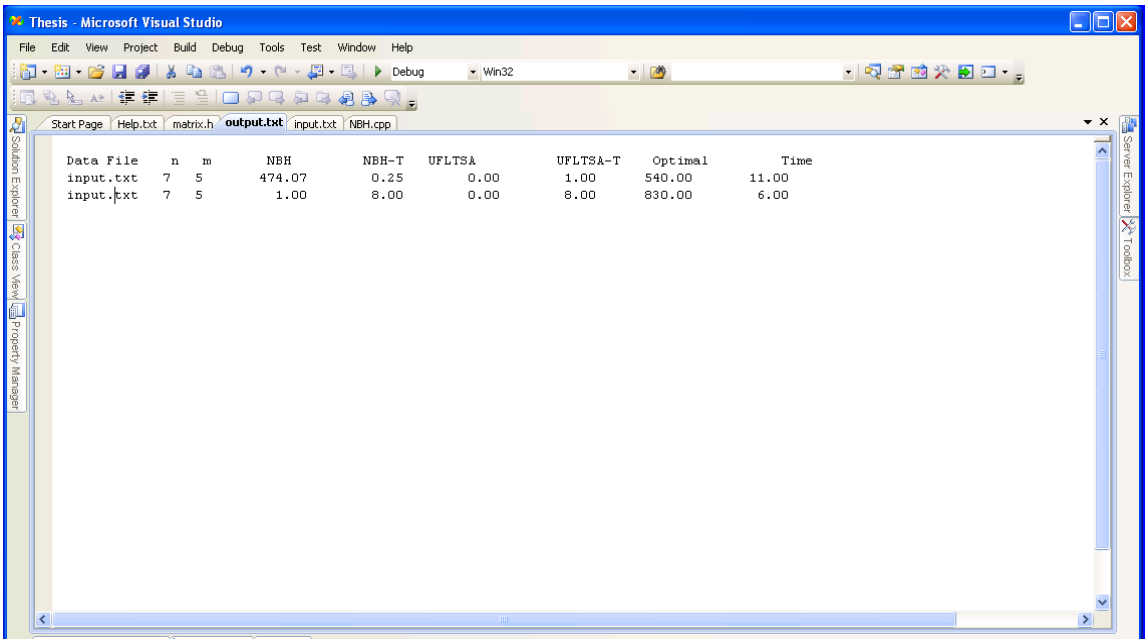


Table6.2: **For two site establishment**

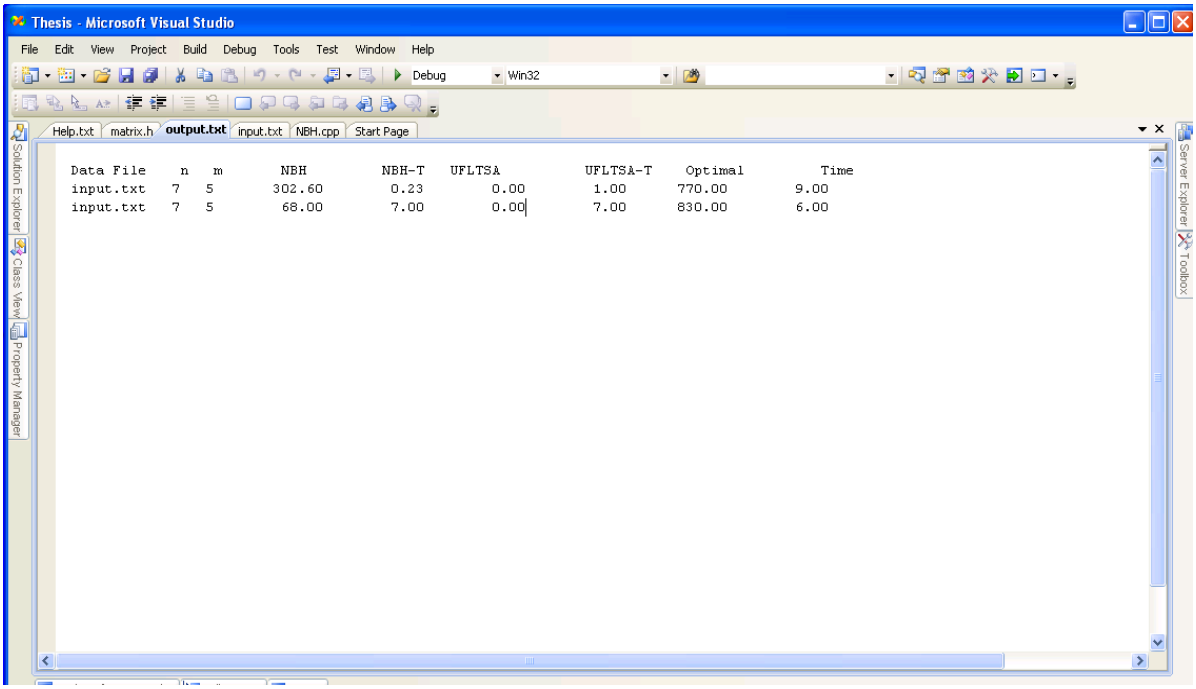


Table6.3: **For three sites establishment.**

Chapter 7
Conclusion And Future Scope

7.1 Conclusion:

A new algorithm for the un-capacitated facility location problem has been presented. The algorithm is based on the tabu search technique. Computational results show that proposed algorithm found the optimal solutions for all problems tested.

Future Scope:

- One can extend the work by investigating other features for generating the random sequences and by using other elements of tabu search such as long-term memory, strategic oscillation, path relinking, among others.
- This algorithm is also extended to solve multi objective constraints.
- One could also improve the performance of the proposed heuristic for getting a good initial solution.

Complexity: Complexity of this algorithm is always $O(n^2)$

References

[1] Glover F.," Future paths for Integer Programming and Links to Artificial intelligence", *Computers and Operations Research*, 5:533-549, 1986. *http://www.upt.pt/tabusearch*

[2] Glover, F. and M. Laguna (1997) *Tabu Search*, Kluwer Academic Publishers, Boston. This is the first comprehensive book on tabu search. The material presented in this article was principally adapted from this book.

[3] R. Eberhart, Y. Shi, and J. Kennedy. Swarm intelligence. Morgan Kaufmann, 2001

[4] Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 191-202 www.stacs-conf.org Shiri Chechik And David Peleg

[5] Prakash, S., Madhusudan, A. and Kunal (2007), "Clustering offices of a company to internet services provider", Proceedings of the National Academy of Sciences, India, Section A, 77(1), 31-35.

[6] Prakash, S. and Gupta, A. (2006), "Nondominated solutions of a site selection problem with two objectives without assignment of priorities to them", News Bulletin of the Calcutta Mathematical Society, 29(1-3), 3-8.

[7] Prakash, S., Balaji, B.V. and Tuteja, Deepak (1999), "Optimizing dead mileage in urban bus routes through a non-dominated solution approach", European Journal of Operational Research, 114, 465-473.

[8] Praveena, V., Prasad, B.V.N.S. and Prakash, S. (1999), "Selection of warehouse sites and clustering ration shops to them with two objectives", Proceedings of the Seminar on Recent Trends and Advances of Mathematics and Statistics in Engineering & Technology held at Indian School of Mines, Dhanbad, India

[9] Rardin, R.L. (1998), "Problem Solving with Mathematical Models" & "Discrete Optimization Methods", In: **Optimization in Operations Research**, Pearson Education (Singapore) Pte. Ltd., Indian Branch, New Delhi, 15-16 and 670-705.

[10] Ignizio, J.P. and Cavalier, T.M. (1994), "Heuristic Programming AND AI", In: **Linear Programming**, Prentice-Hall, Englewood Cliffs, New Jersey, 457-505.

[11] Steuer, R.E. (1986), **Multiple Criteria Optimization: Theory, Computation and Application**, Wiley, New York.

[12] Ignizio, J.P. (1982), "Formulation of the Multi-Objective Model", In: **Linear Programming in Single and Multiple Objective Systems**, Prentice-Hall, Englewood Cliffs, New Jersey, 374-376.

[13] Selection of warehouse sites for clustering ration shops to them with two objectives through a heuristic algorithm incorporating tabu search by Mahesh.k. Sharma

[14] J.A. Bland, G.P. Dawson, Tabu search and design optimization, Computer-Aided Design 23 (1991)195–201.

[15] A.R. Conn, G. Cornuéjols, A projection method for the un-capacitated facility location problem, Mathematical Programming 46(1990)273–298.

[16] G. Cornuéjols, G.L. Nemhauser, L.A. Wolsey, The un-capacitated facility location problem, in: *Discrete Location Theory*, eds. P.B. Mirchandani and R.L. Francis, Wiley, New York, 1990, pp.119–171.

[17] M. Körkel, On the exact solution of large-scale simple plant location problems, European Journal of Operational Research 39(1989)157–173.

[18] J. Krarup, P.M. Pruzan, The simple plant location problem: Survey and synthesis, European Journal of Operational Research 12(1983)36–81.

[19] Narendhar Maaroju, Deepak Garg "*Choosing the Best Heuristic for A NP-Problem*" International Journal of Information Technology & Knowledge Management Issue 2 Vol 1 Year 2008 ISSN 0973-4414.pp 537-547.

[20] R. Battiti. "Reactive search: towards self-tuning heuristics", in Modern heuristic search methods. Wiley&Sons, 1996, pp. 61-83.

[21] J.C. Crput, A. Koukam, T. Lissajoux, A. Caminada. "*Automatic Mesh Generation for Mobile Network Dimensioning Using Evolutionary Approach*", in IEEE Transactions on Evolutionary Computation, vol. 9, no. 1, Feb. 2005, pp. 18–30.

List of Publications

Communicated:

- Bala krishna G, Dr. Deepak Garg, Dr. Mahesh Kumar Sharma, “An approach to solve a bi-objective un-capacitated facility location problem” Journal Of Applied Mathematics & Informatics (JAMI).