Chapter 2
Heuristic Methods

Abstract Since the linear ordering problem is NP-hard, we cannot expect to be able to solve practical problem instances of arbitrary size to optimality. Depending on the size of an instance or depending on the available CPU time we will often have to be satisfied with computing approximate solutions. In addition, under such circumstances, it might be impossible to assess the real quality of approximate solutions. In this and in the following chapter we will deal with the question of how to find very good solutions for the LOP in short or reasonable time. The methods described in this chapter are called *heuristic algorithms* or simply *heuristics*. This term stems from the Greek word *heuriskein* which means to find or discover. It is used in the field of optimization to characterize a certain kind of problem-solving methods. There are a great number and variety of difficult problems, which come up in practice and need to be solved efficiently, and this has promoted the development of efficient procedures in an attempt to find good solutions, even if they are not optimal. These methods, in which the process speed is as important as the quality of the solution obtained, are called heuristics or *approximative algorithms*.

2.1 Introduction

As opposed to *exact methods*, which guarantee to give an optimum solution of the problem, heuristic methods only attempt to yield a good, but not necessarily optimum solution. Nevertheless, the time taken by an exact method to find an optimum solution to a difficult problem, if indeed such a method exists, is in a much greater order of magnitude than the heuristic one (sometimes taking so long that in many cases it is inapplicable). Thus we often resort to heuristic methods to solve real optimization problems.

Perhaps the following comment by Onwubolu and Babu [105] is a little far-fetched: "The days when researchers emphasized using deterministic search techniques to find optimal solutions are gone.". But it is true that in practice an engineer,

R. Martí and G. Reinelt, *The Linear Ordering Problem*, Exact and Heuristic Methods in Combinatorial Optimization 175, DOI: 10.1007/978-3-642-16729-4_2,

an analyst or a manager sometimes might have to make a decision as soon as possible in order to achieve desirable results.

Recent years have witnessed a spectacular growth in the development of heuristic procedures to solve optimization problems. This fact is clearly reflected in the large number of articles published in specialized journals. 1995 saw the first issue of the *Journal of Heuristics*, dedicated solely to the publication of heuristic procedures. In the same year the first international congress dealing with these methods, called the *Metaheuristic International Conference (MIC)*, was held in Breckenridge, Colorado (USA).

In addition to the need to find good solutions of difficult problems in reasonable time, there are other reasons for using heuristic methods, among which we want to highlight:

- No method for solving the problem to optimality is known.
- Although there is an exact method to solve the problem, it cannot be used on the available hardware.
- The heuristic method is more flexible than the exact method, allowing, for example, the incorporation of conditions that are difficult to model.
- The heuristic method is used as part of a global procedure that guarantees to find the optimum solution of a problem.

A good heuristic algorithm should fulfil the following properties:

- A solution can be obtained with reasonable computational effort.
- The solution should be near optimal (with high probability).
- The likelihood for obtaining a bad solution (far from optimal) should be low.

There are many heuristic methods that are very different in nature. Therefore, it is difficult to supply a full classification. Furthermore, many of them have been designed to solve a specific problem without the possibility of generalization or application to other similar problems. The following outline attempts to give wide, non-excluding categories, under which to place the better-known heuristics:

Decomposition Methods

The original problem is broken down into sub-problems that are simpler to solve, bearing in mind, be it in a general way, that subproblems belong to the same problem class.

Inductive Methods

The idea behind these methods is to generalize the smaller or simpler versions to the whole case. Properties or techniques that have been identified in these cases which are easier to analyze, can be applied to the whole problem.

Reduction Methods

These involve identifying properties that are mainly fulfilled by the good solutions and introduce them as boundaries to the problem. The objective is to restrict the space of the solutions by simplifying the problem. The obvious risk is that the optimum solutions of the original problem may be left out.

Constructive Methods

These involve building a solution to the problem literally step by step from scratch. Usually they are deterministic methods and tend to be based on the best choice in each iteration. These methods have been widely used in classic combinatorial optimization.

Local Search Methods

In contrast to the methods previously mentioned, local improvement or local search starts with some feasible solution of the problem and tries to progressively improve it. Each step of the procedure carries out a movement from one solution to another one with a better value. The method terminates when, for a solution, there is no other accessible solution that improves it.

Even though all these methods have contributed to expanding our knowledge of solving real problems, the constructive and local search methods form the foundations of the meta-heuristic procedures [4], which will be described in the next chapter.

2.1.1 Assessing the Quality of Heuristics

There are diverse possibilities for measuring the quality of a heuristic, among which we find the following.

Comparison with the Optimum Solution

Although one normally resorts to an approximative algorithm, because no exact method exists to obtain an optimum solution or it is too time-consuming, sometimes a procedure is available that provides an optimum for a limited set of examples (usually small sized instances). This set of examples can be used to assess the quality of the heuristic method.

Normally, for each example, the following are measured: the percentaged deviation of the heuristic solution value as compared to the optimum one and the mean

of these deviations. If we denote by c_A the value of the solution delivered by heuristic A and by c_{opt} the optimum value of a given example, in a maximization problem like the LOP, the percentaged deviation, *PerDev*, is given by the expression

$$PerDev = 100 \cdot \frac{c_{opt} - c_A}{c_{opt}}.$$

(We assume that all feasible solutions have a positive value.)

Comparison with a Bound

There are situations when no optimum solution is available for a problem, not even for a limited set of examples. An alternative evaluation method involves comparing the value of the solution provided by the heuristic with a bound for the problem (a lower bound if it is a minimization problem and an upper bound if it is a maximization problem). Obviously the quality of fit will depend on the quality of the bound (closeness to optimal). Thus we must somehow have information about the quality of the aforementioned bound, otherwise the proposed comparison would not be of much interest.

Comparison with a Truncated Exact Method

An enumerative method like branch-and-bound explores very many solutions, even though this may be a fraction of the total, and therefore large-scale problems can be computationally out of reach using these methods. Nevertheless, we can establish a limit on the maximum number of iterations (or on the CPU time) to run the exact algorithm. Moreover, we can modify the criteria to fathom a node in the search tree by adding or subtracting (depending on whether it is a minimization or maximization problem) a value Δ to the bound of the node thus fathoming a larger number of nodes and speeding up the method. In this way it guarantees that the value of the best solution provided by the procedure is no further than distance Δ from the optimal value to the problem. In any case, the best solution found with these truncated procedures establishes a bound against which the heuristic can be measured.

Comparison with Other Heuristics

This is one of the most commonly used methods for difficult problems which have been worked on for a long time and for which some good heuristics are known. Similarly to what happens with the bound comparisons, the conclusion of this comparison deals with the quality of fit of the chosen heuristic.

Given that the LOP has been studied in-depth from both the exact viewpoint and that of a heuristic, we have a value of the optimum solution for small and

medium-scale examples, which enables us to establish the optimal deviation in the solution obtained by the heuristics. Furthermore, we can compare the values obtained between the different heuristics to solve the same examples of any size.

Worst Case Analysis

One method that was well-accepted for a time concerns the behavioral analysis of the heuristic algorithm in the worst case; i.e., consider the examples that most disfavor the algorithm and set analytical bounds to the maximal deviation in terms of the optimum solution to the problem. The best aspect of this method is that it established the limits of the algorithm's results for any example. However, for the same reason, the results tend not to be representative of the average behavior of the algorithm. Furthermore, the analysis can be very complicated for more sophisticated heuristics.

An algorithm A for dealing with a maximization problem is called *ε-approximative* if there is a constant $\varepsilon > 0$ such that for every problem instance the algorithm guarantees that a feasible solution can be found with value c_{A} and the property

$$c_{\mathrm{A}} \geq (1 - \varepsilon) c_{\mathrm{opt}}.$$

The analogous definition for minimization problems is $c_{\mathrm{A}} \leq (1 + \varepsilon) c_{\mathrm{opt}}$.

Concerning the approximability of the LOP the following results are known. Suppose that all objective function coefficients are nonnegative and take some arbitrary ordering. Then either this ordering or its reverse version contains at least half of the sum of all coefficients. So $\frac{1}{2}$-approximation of the LOP is trivial, but nothing better is known.

2.2 Construction Heuristics

We will now review some of the construction heuristics, i.e., methods which follow some principle for successively constructing a linear ordering. The principle should somehow reflect that we are searching for an ordering with high value.

2.2.1 The Method of Chenery and Watanabe

One of the earliest heuristic methods was proposed by Chenery and Watanabe [32]. These authors did not formulate an algorithm, but just gave some ideas of how to obtain plausible rankings of the sectors of an input-output table. Their suggestion is to rank those sectors first which show a small share of inputs from other sectors and of outputs to final demand. Sectors having a large share of inputs from other

industries and of final demand output should be ranked last. Chenery and Watanabe defined coefficients taking these ideas into account to find a preliminary ranking. Then they try to improve this ranking in some heuristic way which is not specified in their paper. The authors admit that their method does not necessarily lead to good approximate solutions of the triangulation problem.

2.2.2 Heuristics of Aujac & Masson

This method [6] is based on so-called *output coefficients*. The output coefficient of a sector i with respect to another sector j is defined as

$$b_{ij} = \frac{c_{ij}}{\sum\limits_{k \neq i} c_{ik}} .$$

Then it is intended to rank sector i before sector j whenever $b_{ij} > b_{ji}$ ("better customer principle"). This is impossible in general. So it is heuristically tried to find a linear ordering with few contradictions to this principle. Subsequently local changes are performed to achieve better triangulations. Similarly an *input coefficient* method can be formulated based on the input coefficients

$$a_{ij} = \frac{c_{ij}}{\sum\limits_{k \neq j} c_{kj}} .$$

2.2.3 Heuristics of Becker

In [8] two further methods are described. The first one is related to the previous ones in that it calculates special quotients to rank the sectors. For each sector i the number

$$q_i = \frac{\sum\limits_{k \neq i} c_{ik}}{\sum\limits_{k \neq i} c_{ki}}$$

is determined. The sector with the largest quotient q_i is then ranked highest. Its corresponding rows and columns are deleted from the matrix, and the procedure is applied to the remaining sectors.

Heuristic of Becker (1)

(1) Set $S = \{1,2,\ldots,n\}$.
(2) For $k = 1,2,\ldots,n$:

(2.1) For each $i \in S$ compute $q_i = \frac{\sum_{j \in S \setminus \{i\}} c_{ij}}{\sum_{j \in S \setminus \{i\}} c_{ji}}$.
(2.2) Let $q_j = \max\{q_i \mid i \in S\}$.
(2.3) Set $i_k = j$ and $S = S \setminus \{j\}$.

The second method starts with an arbitrarily chosen linear ordering, w.l.o.g. $\langle 1,2,\ldots,n\rangle$. Then for every $m = 1,2,\ldots,n-1$ the objective function values of the orderings $\langle m+1,m+2,\ldots,n,1,\ldots,m\rangle$ are evaluated. The best one among them is chosen, and the procedure is repeated as long as improvements are possible.

Heuristic of Becker (2)

(1) Generate a random ordering.
(2) Let $\langle i_1,i_2,\ldots,i_n\rangle$ denote the current ordering.
(3) Evaluate all of the orderings $\langle i_{m+1},i_{m+2},\ldots,i_n,1,2,\ldots,i_m\rangle$, for $m = 1,2,\ldots,n-1$.
(4) If the best one among these orderings is better than the current one, take it as the new current ordering and goto (3).

2.2.4 Best Insertion

This is a simple heuristic which builds an ordering by inserting the next objects at positions which are locally optimal.

Best Insertion

(1) Select an arbitrary object j and set $S = \{1,2,\ldots,n\} \setminus \{j\}$. Let $\langle j\rangle$ be the current ordering.
(2) For $k = 1,2,\ldots,n-1$:

(2.1) Let $\langle i_1,i_2,\ldots,i_k\rangle$ denote the current ordering and choose some $l \in S$.
(2.2) For every t, $1 \le t \le k+1$, compute $q_t = \sum_{j=1}^{t-1} c_{i_j l} + \sum_{j=t}^{k} c_{l i_j}$ and let $q_p = \max\{q_t \mid 1 \le t \le k\}$.
(2.3) Insert l at position p in the current ordering and set $S = S \setminus \{l\}$.

An alternative version of step (2.2) computes

$$q_t = \sum_{j=1}^{t-1} c_{i_j l} + \sum_{j=t}^{k} c_{l i_j} - \sum_{j=1}^{t-1} c_{l i_j} - \sum_{j=t}^{k} c_{i_j l}$$

to account for the sum of entries which are "lost" when l is inserted at position t.

Table 2.1 Constructive methods on OPT-I instances

	CW	AM-O	AM-I	Bcq	Bcr	BI1	BI2
IO							
Dev(%)	19.07	32.94	31.45	4.07	30.19	3.24	4.18
Score	231	291	266	101	289	89	104
#Opt	0	0	0	0	0	0	0
SGB							
Dev(%)	12.83	26.15	26.15	3.57	31.56	3.89	3.03
Score	100	125	125	54	175	56	40
#Opt	0	0	0	0	0	0	0
RandomAII							
Dev(%)	2.60	36.50	36.55	1.57	37.75	1.09	1.26
Score	100	135	136	68	162	34	48
#Opt	0	0	0	0	0	0	0
RandomB							
Dev(%)	10.13	24.69	24.69	7.04	26.41	5.24	4.87
Score	276	368	368	194	454	124	106
#Opt	0	0	0	0	0	0	0
MB							
Dev(%)	8.40	43.37	43.37	2.90	40.30	2.49	2.27
Score	120	178	178	80	154	52	48
#Opt	0	0	0	0	0	0	0
Special							
Dev(%)	0.02	0.57	0.14	3.10	0.40	0.01	0.17
Score	64	178	113	210	149	41	83
#Opt	0	0	0	0	0	4	3
OPT-I							
Avg. Dev(%)	10.85	32.97	32.55	3.95	32.35	3.49	3.50
Sum #Opt	0	0	0	0	0	4	3

Table 2.1 reports on our results for 7 constructive heuristics on the OPT-I set (the set of 229 instances with optimum known). In this experiment we compute for each instance and each method the relative deviation *Dev* (in percent) between the best solution value *Value* obtained with the method and the optimal value for that instance. For each method, we also report the number of instances *#Opt* for which an optimum solution could be found. In addition, we calculate the so-called

score statistic [114] associated with each method. For each instance, the *nrank* of method *M* is defined as the number of methods that produce a better solution than the one found by *M*. In the event of ties, the methods receive the same *nrank*, equal to the number of methods strictly better than all of them. The value of *Score* is the sum of the *nrank* values for all the instances in the experiment. Thus the lower the *Score* the better the method. We do not report running times in this table because these methods are very fast and their running times are extremely short (below 1 millisecond). Specifically, Table 2.1 shows results for:

- `CW`: Chenery and Watanabe algorithm
- `AM-O`: Aujac and Masson algorithm (output coefficients)
- `AM-I`: Aujac and Masson algorithm (input coefficients)
- `Bcq`: Becker algorithm (based on quotients)
- `Bcr`: Becker algorithm (based on rotations)
- `BI1`: Best Insertion algorithm (variant 1)
- `BI2`: Best Insertion algorithm (variant 2)

Results in Table 2.1 clearly indicate that OPT-I instances pose a challenge for the simple heuristics with average percentage deviations ranging from 3.49% to 32.97%. In most of the cases none of the methods is able to match the optimum solution (with the exception of `BI1` and `BI2` with 4 and 3 optima respectively in the Special instances). These results show that only `Bcq`, `BI1` and `BI2` can be considered reasonable construction heuristics (with an average percent deviation lower than 5%).

2.3 Local Search

After having constructed some ordering with one of the heuristics above, it is reasonable to look for improvement possibilities. In this section we will describe fairly simple (deterministic) local improvement methods that are able to produce acceptable solutions for the LOP. The basic philosophy that drives local search is that it is often possible to find a good solution by repeatedly increasing the quality of a given solution, making small changes at a time called *moves*. The different types of possible moves characterize the various heuristics. Starting from a solution generated by a construction heuristic, a typical local search performs steps as long as the objective function increases.

Local search can only be expected to obtain optimum or near-optimum solutions for easy problems of medium size, but it is a very important and powerful concept for the design of meta-heuristics, which are the topic of the next chapter.

2.3.1 Insertion

This heuristic checks whether the objective function can be improved if the position of an object in the current ordering is changed. All possibilities for altering the position of an object are checked and the method stops when no further improvement is possible this way.

In problems where solutions are represented as permutations, insertions are probably the most direct and efficient way to modify a solution. Note that other movements, such as swaps, can be obtained by composition of two or more insertions. We define $move(O_j, i)$ as the modification which deletes O_j from its current position j in permutation O and inserts it at position i (i.e., between the objects currently in positions $i-1$ and i).

Now, the insertion heuristic tries to find improving moves examining eventually all possible new positions for all objects O_j in the current permutation O. There are several ways for organizing the search for improving moves. For our experiments we proceeded as follows:

Insertion

(1) Compute an initial permutation $O = \langle O_1, O_2, \ldots, O_n \rangle$.
(2) For $j = 1, 2, \ldots, n$:
  (2.1) Evaluate all possible insertions $move(O_j, i)$.
  (2.2) Let $move(O_k, i^*)$ be the best of these moves.
  (2.3) If $move(O_k, i^*)$ is improving then perform it and update O.
(3) If some improving move was found, then goto (2).

In [86] two neighborhoods are studied in the context of local search methods for the LOP. The first one consists of permutations obtained by switching the positions of contiguous objects O_j and O_{j+1}. The second one involves all permutations resulting from executing general insertion moves, as defined above. The conclusion from the experiments is that the second neighborhood clearly outperforms the first one, which is much more limited. Furthermore two strategies for exploring the neighborhood of a solution were studied. The *best* strategy selects the move with the largest *move value* among all the moves in the neighborhood. The *first* strategy, on the other hand, scans the list of objects (in the order given by the current permutation) searching for the first object whose movement gives a strictly positive move value. The computations revealed that both strategies provide similar results but the *first* involved lower running times.

2.3.2 The Heuristic of Chanas & Kobylanski

The method developed by Chanas and Kobylanski [32], referred to as the CK method in the following, is based on the following symmetry property of the LOP. If the permutation $O = \langle O_1, O_2, \ldots, O_n \rangle$ is an optimum solution to the maximization problem, then an optimum solution to the minimization problem is $O^* = \langle O_n, O_{n-1}, \ldots, O_1 \rangle$. In other words, when the sum of the elements above the main diagonal is maximized, the sum of the elements below the diagonal is minimized. The CK method utilizes this property to escape local optimality. In particular, once a local optimum solution O is found, the process is re-started from the permutation O^*. This is called the REVERSE operation.

In a global iteration, the CK method performs insertions as long as the solution improves. Given a solution, the algorithm explores the insertion move $move(O_j, i)$ of each element O_j in all the positions i in O, and performs the best one. When no further improvement is possible, it generates a new solution by applying the REVERSE operation from the last solution obtained, and performs a new global iteration. The method finishes when the best solution found cannot be improved upon in the current global iteration.

It should be noted that the CK method can be considered to be a generalization of the second heuristic of Becker described above. The latter evaluates the orderings that can be obtained by rotations of a solution, while the CK method evaluates all insertions. Since these rotations are basically insertions of the first elements to the last positions, we can conclude that Becker's method explores only a fraction of the solutions explored by CK.

2.3.3 *k*-opt

The k-opt improvement follows a principle that can be applied to many combinatorial optimization problems. Basically, it selects k elements of a solution and locally optimizes with respect to these elements. For the LOP, a possible k-opt heuristic would be to consider all subsets of k objects $O_{i_1}, \ldots, O_{i_k}$ in the current permutation and find the best assignment of these objects to the positions $i_1, \ldots, i_k$. Since the number of possible new assignments grows exponentially with k, we have only implemented 2-opt and 3-opt.

2.3.4 Kernighan-Lin Type Improvement

The main problem with local improvement heuristics is that they very quickly get trapped in a local optimum. Kernighan and Lin proposed the idea (originally in [78] for a partitioning problem) of looking for more complicated moves that are composed of simpler moves. In contrast to pure improvement heuristics, it allows that

some of the simple moves are not improving. In this way the objective can decrease locally, but new possibilities arise for escaping from the local optimum. This type of heuristic proved particularly effective for the traveling salesman problem (where it is usually named *Lin-Kernighan heuristic*).

We only describe the principle of the Kernighan-Lin approach. For practical applications on large problems, it has to be implemented carefully with appropriate data structures and further enhancements like restricted search or limited length of combined moves to speed up the search for improving moves. We do not elaborate on this here.

We devised two heuristics of this type for the LOP. In the first version, the basic move consists of interchanging two objects in the current permutation.

Kernighan-Lin 1

(1) Compute some linear ordering O.
(2) Let $m = 1$, $S_m = \{1,2,\ldots,n\}$.
(3) Determine objects $s,t \in S_m$, $s \neq t$, the interchange of which in the current ordering leads to the largest increase g_m of the objective function (increase may be negative).
(4) Interchange s and t in the current ordering. Set $s_m = s$ and $t_m = t$.
(5) If $m < \lfloor n/2 \rfloor$, set $S_{m+1} = S_m \setminus \{s,t\}$ and $m = m+1$. Goto (3).
(6) Determine $1 \leq k \leq m$, such $G = \sum_{i=1}^{k} g_i$ is maximum.
(7) If $G \leq 0$ then `Stop`, otherwise, starting from the original ordering O, successively interchange s_i and t_i, for $i = 1,2,\ldots,k$. Let O denote the new ordering and goto (2).

The second version builds upon insertion moves.

Kernighan-Lin 2

(1) Compute some linear ordering O.
(2) Let $m = 1$, $S_m = \{1,2,\ldots,n\}$.
(3) Among all possibilities for inserting an object of S_m at a new position determine the one leading to the largest increase g_p of the objective function (increase may be negative). Let s be this object and p the new position.
(4) Move s to position p in the current ordering. Set $s_m = s$ and $p_m = p$.
(5) If $m < n$, set $S_{m+1} = S_m \setminus \{s\}$ and $m = m+1$. Goto (3).
(6) Determine $1 \leq k \leq m$, such $G = \sum_{i=1}^{k} g_i$ is maximum.
(7) If $G \leq 0$ then `Stop`, otherwise, starting from the original ordering O, successively move s_i to position p_i, for $i = 1,2,\ldots,k$. Let O denote the new ordering and goto (2).

2.3.5 Local Enumeration

This heuristic chooses windows $\langle i_k, i_{k+1}, \ldots, i_{k+L-1} \rangle$ of a given length L of the current ordering $\langle i_1, i_2, \ldots, i_n \rangle$ and determines the optimum subsequence of the respective objects by enumerating all possible orderings. The window is moved along the complete sequence until no more improvements can be found. Of course, L cannot be chosen too large because the enumeration needs time $O(L!)$.

Local Enumeration

(1) Compute some linear ordering O.
(2) For $i = 1, \ldots, n - L + 1$:
  (2.1) Find the best possible rearrangement of the objects at positions $i, i+1, \ldots, i+L-1$.
(3) If an improving move has been found in the previous loop, then goto (2).

Table 2.2 reports on our results for 7 improving heuristics on the OPT-I set of instances. As in the construction heuristics, we report, for each instance and each method, the relative percent deviation *Dev*, the number of instances *#Opt* for which an optimum solution is found, and the *score* statistic. Similarly, we do not report running times in this table because these methods are fairly fast. Specifically, the results obtained with the following improvement methods (started with a random initial solution) are given:

- `LSi`: Local Search based on insertions
- `2opt`: Local Search based on 2-opt
- `3opt`: Local Search based on 3-opt
- `LSe`: Local Search based on exchanges
- `KL1`: Kernighan-Lin based on exchanges
- `KL2`: Kernighan-Lin based on insertions
- `LE`: Local enumeration

As expected, the improvement methods are able to obtain better solutions than the construction heuristics, with average percentage deviations (shown in Table 2.2) ranging from 0.57% to 2.30% (the average percentage deviations of the construction heuristics range from 3.49% to 32.97% as reported in Table 2.1). We have not observed significant differences when applying the improvement method from different initial solutions. For example, as shown in Table 2.2 the LSi method exhibits a *Dev* value of 0.16% on the RandomAII instances when it is started from random solutions. When it is run from the CW or the Bcr solutions, it obtains a *Dev* value of 0.17% and 0.18% respectively.

Table 2.2 Improvement methods on OPT-I instances

	LSi	2opt	3opt	LSe	KL1	KL2	LE
IO							
Dev(%)	1.08	0.64	0.23	1.73	1.35	4.24	0.01
Score	243	181	125	295	239	232	49
#Opt	0	1	4	0	1	0	43
SGB							
Dev(%)	0.16	0.81	0.53	1.35	0.63	0.28	1.09
Score	42	122	84	154	100	63	135
#Opt	1	0	0	0	0		0
RandomAII							
Dev(%)	0.16	0.77	0.38	0.62	0.61	0.09	0.54
Score	46	161	81	134	134	29	112
#Opt	0	0	0	0	0	0	0
RandomB							
Dev(%)	0.79	4.04	2.13	3.78	3.51	0.61	3.56
Score	124	400	232	387	359	95	362
#Opt	1	0	0	0	0	1	0
MB							
Dev(%)	0.02	0.57	0.14	3.10	0.40	0.01	0.17
Score	64	178	113	210	149	41	83
#Opt	0	0	0	0	0	4	3
Special							
Dev(%)	1.19	3.30	2.05	3.21	2.40	0.89	3.52
Score	69	144	82	138	120	49	156
#Opt	4	2	2	2	3	3	3
OPT-I							
Avg. Dev(%)	0.57	1.69	0.91	2.30	1.49	1.02	1.48
Sum #Opt	5	3	6	2	4	8	49

2.4 Multi-Start Procedures

Multi-start procedures were originally conceived as a way of exploiting a local or neighborhood search procedure, by simply applying it from multiple random initial solutions. It is well known that search methods based on local optimization, aspiring to find global optima, usually require certain diversification to overcome local optimality. Without this diversification, such methods can become reduced to tracing paths that are confined to a small area of the solution space, making it impossible to find a global optimum. *Multi-start algorithms* can be considered to be a bridge between simple (classical) heuristics and complex (modern) meta-heuristics. The *re-start mechanism* of multi-start methods can be super-imposed on many different search methods. Once a new solution has been generated, a variety of options can be used to improve it, ranging from a simple greedy routine to a complex meta-heuristic. This section focuses on the different strategies and methods that can be

used to generate solutions to launch a succession of new searches for a global optimum.

The principle layout of a multi-start procedure is the following.

Multi-Start

(1) Set i=1.
(2) While the stopping condition is not satisfied:
  (2.1) Construct a solution x_i. **(Generation)**
  (2.2) Apply local search to improve x_i and let x_i' be the solution obtained. **(Improvement)**
  (2.3) If x_i' improves the best solution, update it. Set $i = i + 1$. **(Test)**

The computation of x_i in (2.1) is typically performed with a constructive algorithm. Step (2.2) tries to improve this solution, obtaining x_i'. Here, a simple improvement method can be applied. However, this second phase has recently become more elaborate and, in some cases, is performed with a complex meta-heuristic that may or may not improve the initial solution x_i (in this latter case we set $x_i' = x_i$).

2.4.1 Variants of Multi-Start

We will first review some relevant contributions on multi-start procedures.

Early papers on multi-start methods are devoted to the Monte Carlo random restart in the context of nonlinear unconstrained optimization, where the method simply evaluates the objective function at randomly generated points. The probability of success approaches 1 as the sample size tends to infinity under very mild assumptions about the objective function. Many algorithms have been proposed that combine the Monte Carlo method with local search procedures [115]. The convergence for random re-start methods is studied in [120], where the probability distribution used to choose the next starting point can depend on how the search evolves. Some extensions of these methods seek to reduce the number of complete local searches that are performed and increase the probability that they start from points close to the global optimum [96].

In [13] relationships among local minima from the perspective of the best local minimum are analyzed, finding convex structures in the cost surfaces. Based on the results of that study, they propose a multi-start method where starting points for greedy descent are adaptively derived from the best previously found local minima. In the first step, *adaptive multi-start heuristics* generate random starting solutions and run a greedy descent method from each one to determine a set of corresponding random local minima. In the second step, *adaptive starting solutions* are constructed based on the local minima obtained so far and improved with a greedy descent method. This improvement is applied several times from each adaptive starting

solution to yield corresponding *adaptive local minima*. The authors test this method for the traveling salesman problem and obtain significant speedups over previous multi-start implementations.

Simple forms of multi-start methods are often used to compare other methods and measure their relative contribution. In [7] different genetic algorithms for six sets of benchmark problems commonly found in the genetic algorithms literature are compared: traveling salesman problem, job-shop scheduling, knapsack and bin packing problem, neural network weight optimization, and numerical function optimization. The author uses the *multi-start method* (*multiple restart stochastic hill-climbing*) as a basis for computational testing. Since solutions are represented with strings, the improvement step consists of a local search based on random flipping of bits. The results indicate that using genetic algorithms for the optimization of static functions does not yield a benefit, in terms of the final answer obtained, over simpler optimization heuristics.

One of the most well known multi-start methods is the *greedy adaptive search procedure (GRASP)*. The GRASP methodology was introduced by Feo and Resende [45] and was first used to solve set covering problems [44]. We will devote a section in the next chapter to describe this methodology in detail.

A multi-start algorithm for unconstrained global optimization based on *quasi-random samples* is presented in [67]. Quasi-random samples are sets of deterministic points, as opposed to random, that are evenly distributed over a set. The algorithm applies an inexpensive local search (steepest descent) on a set of quasi-random points to concentrate the sample. The sample is reduced, replacing worse points with new quasi-random points. Any point that is retained for a certain number of iterations is used to start an efficient complete local search. The algorithm terminates when no new local minimum is found after several iterations. An experimental comparison shows that the method performs favorably with respect to other global optimization procedures.

An open question in order to design a good search procedure is whether it is better to implement a simple improving method that allows a great number of global iterations or, alternatively, to apply a complex routine that significantly improves a few generated solutions. A simple procedure depends heavily on the initial solution but a more elaborate method takes much more running time and therefore can only be applied a few times, thus reducing the sampling of the solution space. Some meta-heuristics, such as GRASP, launch limited local searches from numerous constructions (i.e., starting points). In other methods, such as tabu search, the search starts from one initial point and, if a restarting procedure is also part of the method, it is invoked only a limited number of times. In [94] the balance between restarting and search-depth (i.e., the time spent searching from a single starting point) is studied in the context of the matrix bandwidth problem. Both alternatives were tested with the conclusion that it was better to invest the time searching from a few starting points than re-starting the search more often. Although we cannot draw a general conclusion from these experiments, the experience in the current context and in previous projects indicates that some meta-heuristics, like tabu search, need

to reach a critical search depth to be effective. If this search depth is not reached, the effectiveness of the method is severely compromised.

2.4.2 Experiments with the LOP

In this section we will describe and compare 10 different constructive methods for the LOP. It should be noted that, if a constructive method is completely deterministic (with no random elements), its replication (running it several times) will always produce the same solution. Therefore, we should add random selections in a constructive method to obtain different solutions when replicated. Alternatively, we can modify selections from one construction to another in a deterministic way by recording and using some frequency information. We will look at both approaches, which will enable us to design constructive methods for the LOP that can be embedded in a multi-start procedure.

Above we have described the construction heuristic of Becker [8] in which for each object i the value q_i is computed. Then, the objects are ranked according to the q-values $q_i = \sum_{k \neq i} c_{ik} / \sum_{k \neq i} c_{ki}$.

We now compute two other values that can also be used to measure the attractiveness of an object to be ranked first. Specifically, r_i and c_i are, respectively, the sum of the elements in the row corresponding to object i, and the sum of the elements in the column of object i, i.e., $r_i = \sum_{k \neq i} c_{ik}$ and $c_i = \sum_{k \neq i} c_{ki}$.

Constructive Method G1

This method first computes the r_i values for all objects. Then, instead of selecting the object with the largest r-value, it creates a list with the most attractive objects, according to the r-values, and randomly selects one among them. The selected object is placed first and the process is repeated for n iterations. At each iteration the r-values are updated to reflect previous selections (i.e., we sum the c_{ik} across the unselected elements) and the candidate list for selection is computed with the highest evaluated objects. The method combines the *random selection* with the *greedy evaluation*, and the size of the candidate list determines the relative contribution of these two elements.

Constructive method G1

(1) Set $S = \{1,2,\ldots,n\}$. Let $\alpha \in [0,1]$ be the percentage for selection and O be the empty ordering.
(2) For $t = 1,2,\ldots,n$:
  (2.1) Compute $r_i = \sum_{k \in S, k \neq i} c_{ik}$ for all $i \in S$.
  (2.2) Let $r^* = \max\{r_i \mid i \in S\}$.
  (2.3) Compute the candidate list $C = \{i \in S \mid r_i \geq \alpha r^*\}$.
  (2.4) Randomly select $j^* \in C$ and place j^* at position t in O and set $S = S \setminus \{j^*\}$.

Constructive Methods G2 and G3

Method G2 is based on the c_i-values computed above. It works in the same way as G1 but the attractiveness of object i is now measured with c_i instead of r_i. Objects with large c-values are placed now in the last positions.

Constructive method G2

(1) Set $S = \{1,2,\ldots,n\}$. Let $\alpha \in [0,1]$ be the percentage for selection and O be the empty ordering.
(2) For $t = n, n-1, \ldots, 1$:
  (2.1) Compute $c_i = \sum_{k \in S, k \neq i} c_{ki}$ for all $i \in S$.
  (2.2) Let $c^* = \max\{c_i \mid i \in S\}$.
  (2.3) Compute the candidate list $C = \{i \in S \mid c_i \geq \alpha c^*\}$.
  (2.4) Randomly select $j^* \in C$ and place j^* in position t in O and set $S = S \setminus \{j^*\}$.

In a similar way, constructive method G3 measures the attractiveness of object i for selection with q_i and performs the same steps as G1. Specifically, at each iteration the q-values are computed with respect to the unselected objects, a restricted candidate list is formed with the objects with largest q-values, and one of them is randomly selected and placed first.

Constructive Methods G4, G5 and G6

These methods are designed analogously to G1–G3, except that the selection of objects is from a candidate list of the least attractive and the solution is constructed starting from the last position of the permutation. We give the specification of G6 which is modification of G3.

Constructive method G6

(1) Set $S = \{1,2,\ldots,n\}$. Let $\alpha \geq 0$ be the percentage for selection and O be the empty ordering.
(2) For $t = 1,2,\ldots,n$:

(2.1) For all $i \in S$, compute

$$q_i = \frac{\sum_{k \in S, k \neq i} c_{ik}}{\sum_{k \in S, k \neq i} c_{ki}}.$$

(2.2) Let $q^* = \min\{q_i \mid i \in S\}$.
(2.3) Compute the candidate list $C = \{i \in S \mid q_i \leq (1+\alpha)q^*\}$.
(2.4) Randomly select $j^* \in C$ and place j^* in position $n-t+1$ in O and set $S = S \setminus \{j^*\}$.

Constructive Method MIX

This is a mixed procedure derived from the previous six. The procedure generates a fraction of solutions from each of the previous six methods and combines these solutions into a single set. That is, if n solutions are required, then each method Gi, $i = 1,\ldots,6$, contributes $n/6$ solutions.

Constructive Method RND

This is a random generator. This method simply generates random permutations. We use it as a basis for our comparisons.

Constructive Method DG

This is a general purpose diversification generator suggested in [55] which generates diversified permutations in a systematic way without reference to the objective function.

Constructive Method FQ

This method implements an algorithm with frequency-based memory, as proposed in tabu search [52] (we will see this methodology in the next chapter). It is based on modifying a measure of attractiveness with a frequency measure that discourages

objects from occupying positions that they have frequently occupied in previous solution generations.

The constructive method FQ (proposed in [19]) is based on the notion of constructing solutions employing modified *frequenciesfrequency*. The generator exploits the permutation structure of a linear ordering. A frequency counter is maintained to record the number of times an element i appears in position j. The frequency counters are used to penalize the "attractiveness" of an element with respect to a given position. To illustrate this, suppose that the generator has created 30 solutions. If 20 out of the 30 solutions have element 3 in position 5, then the frequency counter $freq(3,5) = 20$. This frequency value is used to bias the potential assignment of element 3 in position 5 during subsequent constructions, thus inducing *diversification* with respect to the solutions already generated.

The attractiveness of assigning object i to position j is given by the greedy function $fq(i,j)$, which modifies the value of q_i to reflect previous assignments of object i to position j, as follows:

$$fq(i,j) = \frac{\sum\limits_{k \neq i} c_{ik}}{\sum\limits_{k \neq i} c_{ki}} - \beta \frac{\max_q}{\max_f} freq(i,j),$$

where $\max_f = \max\{freq(i,j) \mid i = 1,\ldots,n, j = 1,\ldots,n\}$ and $\max_q = \max\{q_i \mid i = 1,\ldots,n\}$.

Constructive method FQ

(1) Set $S = \{1,2,\ldots,n\}$. Let $\beta \in [0,1]$ be the percentage for diversification and $freq(i,j)$ be the number of times object i has been assigned to position j in previous constructions.
(2) For $t = 1,2,\ldots,n$:

(2.1) For all $i,j \in S$ compute $fq(i,j) = \frac{\sum\limits_{k \neq i} c_{ik}}{\sum\limits_{k \neq i} c_{ki}} - \beta \frac{\max_q}{\max_f} freq(i,j)$.
(2.2) Let i^* and j^* be such that $fq(i^*,j^*) = \max\{fq(i,j) \mid i,j \in S\}$.
(2.3) Place i^* at position j^* in O and set $S = S \setminus \{i^*\}$.
(2.4) $freq(i^*,j^*) = freq(i^*,j^*) + 1$.

It is important to point out that $fq(i,j)$ is an adaptive function since its value depends on attributes of the unassigned elements at each iteration of the construction procedure.

In our first experiment we use the instance `stabu75` from `LOLIB`. We have generated a set of 100 solutions with each of the 10 generation methods. Figure 2.1 shows in a box-and-whisker-plot representation, the value of the 100 solutions generated with each method. Since the LOP is a maximization problem, it is clear that the higher the value, the better the method. We can therefore say that constructive method G3 obtains the best results. Other methods, such as FQ and MIX also obtain

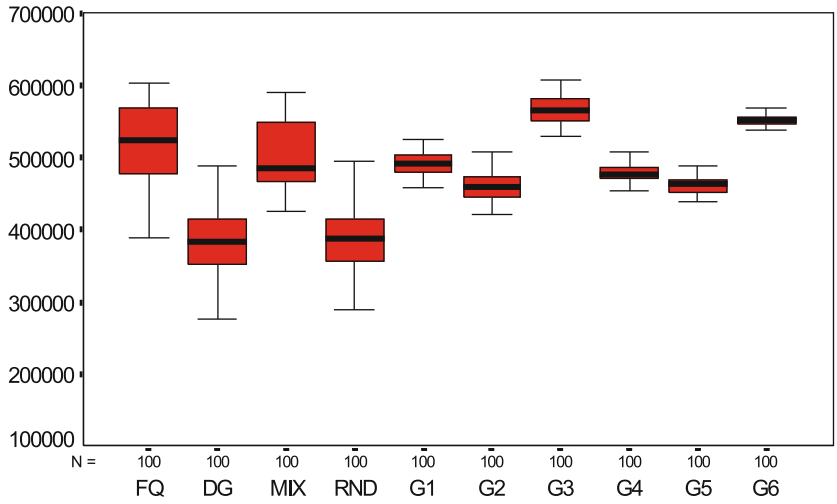

Fig. 2.1 Objective function value box-plot for each method

solutions with very good values, but their box-plot representation indicates that they also produce lower quality solutions. However, if the construction is part of a global method (as is the case in multi-start methods), we may prefer a constructive method able to obtain solutions with different structures rather than a constructive method that provides very similar solutions. Note that if every solution is subjected to local search, then it is preferable to generate solutions scattered in the search space as starting points for the local search phase rather than good solutions concentrated in the same area of the solution space. Therefore, we need to establish a trade off between *quality* and *diversity* when selecting our construction method.

Given a set of solutions P represented as permutations, in [95] a diversity measure d is proposed which consists of computing the distances between each solution and a "center" of P. The sum (or alternatively the average) of these $|P|$ distances provides a measure of the diversity of P. The diversity measure d is calculated as follows:

(1) Calculate the median position of each element i in the solutions in P.
(2) Calculate the *dissimilarity* (distance) of each solution in the population with respect to the median solution. The dissimilarity is calculated as the sum of the absolute difference between the position of the elements in the solution under consideration and the median solution.
(3) Calculate d as the sum of all the individual dissimilarities.

For example, assume that P consists of the orderings $\langle A,B,C,D\rangle$, $\langle B,D,C,A\rangle$, and $\langle C,B,A,D\rangle$. The median position of element A is therefore 3, since it occupies positions 1, 3 and 4 in the given orderings. In the same way, the median positions of B,C and D are 2, 3 and 4, respectively. Note that the median positions might not

induce an ordering, as in the case of this example. The diversity value of the first solution is then calculated as $d_1 = |1-3|+|2-2|+|3-3|+|4-4| = 2$.

In the same way, the diversity values of the other two solutions are obtained as $d_2 = 4$ and $d_3 = 2$. The diversity measure d of P is then given by $d = 2+4+2 = 8$.

We then continue with our experiment to compare the different constructive methods for the LOP. As described above, we have generated a set of 100 solutions with each of the 10 generation methods. Figure 2.2 shows the box-and-whisker plot of the diversity values of the solution set obtained with each method.

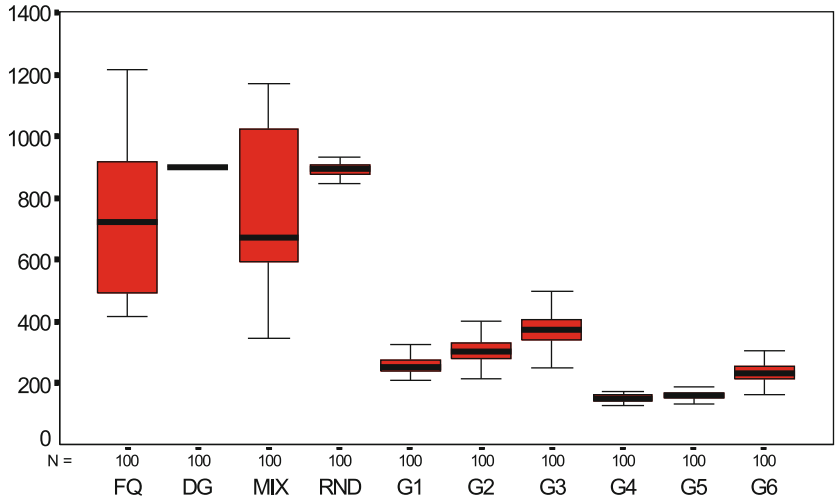

Fig. 2.2 Diversity value box-plot for each method

Figure 2.2 shows that MIX and FQ obtain the highest diversity values (but also generate other solutions with low diversity values). As expected, the random constructive method RND consistently produces high diversity values (always generating solutions with an associated d-value over 800 in the diagram).

As mentioned, a good method must produce a set of solutions with high quality and high diversity. If we compare, for example, generators MIX and G3 we observe in Fig. 2.1 that G3 produces slightly better solutions in terms of solution quality, but Fig. 2.2 shows that MIX outperforms G3 in terms of diversity. Therefore, we will probably consider MIX as a better method than G3. In order to rank the methods we have computed the average of both measures across each set.

Figure 2.3 shows the average of the diversity values on the x-axis and the average of the quality on the y-axis. A point is plotted for each method.

As expected, the random generator RND produces a high diversity value (as measured by the dissimilarity) but a low quality value. DG matches the diversity of RND using a systematic approach instead of randomness, but as it does not use the value of solutions, it also presents a low quality score. The mixed method MIX provides a

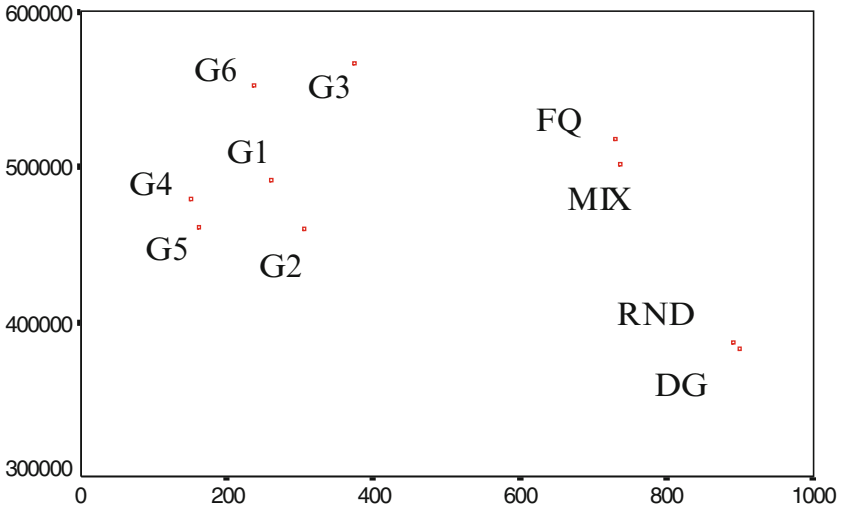

Fig. 2.3 Quality and diversity for each method

good balance between dissimilarity and quality, by uniting solutions generated with methods G1 to G6.

We think that quality and diversity are equally important, so we have added both averages. To do so, we use two relative measures $\triangle C$ for quality, and $\triangle d$ for diversity. They are basically standardizations to translate the average of the objective function values and diversity values respectively to the [0,1] interval. In this way we can simply add both quantities.

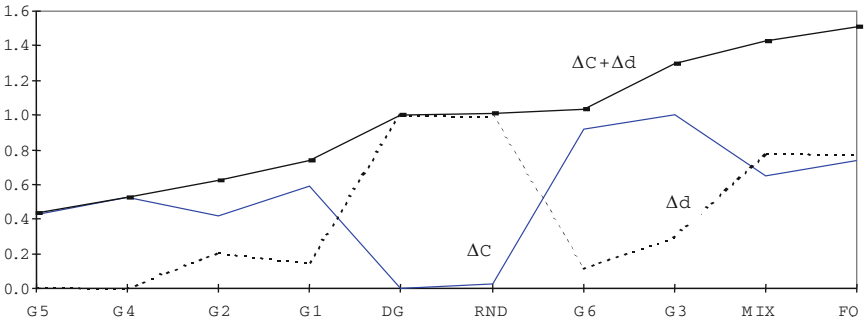

Fig. 2.4 Quality and Diversity for each method

Figure 2.4 clearly shows the following ranking of the 10 methods, where the overall best is the FQ generator: G5, G4, G2, G1, DG, RND, G6, G3, MIX and FQ. These results are in line with previous works which show the inclusion of memory

structures (frequency information) to be effective within the multi-start framework. However, one should note that this method of ranking has been obtained considering both quality and diversity with equal weight. If we vary this criterion, the ranking would also change.

It should be noted that, unlike other well-known methods that we will review in the next chapter, multi-start procedures have not yet become widely implemented and tested as a meta-heuristic itself for solving complex optimization problems. We have shown new ideas that have recently emerged within the multi-start area that add a clear potential to this framework which has yet to be fully explored.

http://www.springer.com/978-3-642-16728-7

The Linear Ordering Problem
Exact and Heuristic Methods in Combinatorial Optimization
Martí, R.; Reinelt, G.
2011, XII, 172 p., Hardcover
ISBN: 978-3-642-16728-7