1

Introduction

1.1 Introduction

Many real-life applications can be modelled reasonably well and solved optimally by one of the classical optimisation techniques familiar to the Operational Research/Management Science/Computer Science/Engineering communities such as linear programming, integer programming, non-linear programming, dynamic programming (DP) and network-based methods, among others. However, there are a lot of applications where the combinatorial effect of the problem makes the determination of the optimal solution intractable, and hence these standard optimal (or exact) techniques become unsuitable. This is because the computer time needed to find such a solution could be too large to be acceptable in real life or the solution found is not really the overall best (global optimum) but just one of the local optima that can be relatively poor when compared to the overall (or global) best one. To overcome such a drawback, heuristic methods were devised with the aim to provide the user with reasonably good solutions, if not the best.

Although heuristic methods do not guarantee optimality, in some situations, they seem to be the only way forward to produce concrete

S. Salhi, *Heuristic Search*, DOI 10.1007/978-3-319-49355-8_1

results. To date, heuristic search methods have been widely used in business, economic, sports, environment, statistics, medicine and engineering problems that were found difficult to solve otherwise. The view of adopting heuristics to solve approximately a large spectrum of complex optimisation problems that were not possible to solve before is now much more accepted and welcomed. There are however measures of performance to assess these heuristics which I shall briefly discuss later in this chapter.

The word 'heuristic' originates from a Greek word that means discover and explore in the wider sense. Heuristics are also known as approximate techniques. The main goal in heuristic search is to construct a model that can be easily understood and that provides *good* solutions in a *reasonable* amount of computing time. Such techniques consist of a combination of scientific components such as mathematical logic, statistics and computing, as well as human factors such as experience, and also in many cases a good insight of the problem that needs to be addressed. The latter may give the impression that such research development can be relatively restrictive when compared to the general techniques, but in my view, this is a crucial component in the design of a heuristic as this could make the technique much faster and more relevant to the problem under study. In certain cases, this can even lead to new ideas that would not have been thought about otherwise. These new ideas can sometimes be translated into formal rules and algorithms that can be used for a wider class of related applications and, hence, could be very rewarding.

In the remainder of this chapter, I first give a brief methodology on how to approach a real problem followed by some performance measures for the evaluation of a given method, and then I describe the proposed simple heuristic categorisation. The main focus of global search is to approximately try to avoid being trapped into a poor local optimum. This is performed by either (a) accepting only improving solutions (Chap. 2), (b) accepting certain uphill moves (Chap. 3) or (c) providing several solutions at once which when combined somehow may generate some new better solutions (Chap. 4). The recent area of hybridisation between heuristics or between heuristics and exact methods is discussed in Chap. 5 followed by some implementation issues in Chap. 6. The final chapter covers real-life applications as well as those applications commonly used

by academics, followed by the last section that summarises the conclusions and some of the research avenues that I believe to be challenging.

1.2 The Optimisation Problem

A general optimisation problem for the case of minimisation can be defined in the following form:

$$(P): \begin{cases} \text{Minimise} & F(X) \\ st & X \in S, S \subseteq E \end{cases}$$

In many applications, (P) can be rather complex to solve because of the following:

(i) The solution space E can be either a finite and very large set rendering (P) as combinatorial optimisation problem, or $E = \mathbb{N}^n$ making (P) an integer optimisation problem, or $E = \mathbb{R}^n$ a continuous optimisation problem.
(ii) The decision variables, X, can be continuous, integer, binary or any combination of the above types.
(iii) The objective function, $F(.)$, may not be necessarily linear, continuous or even convex. $F(.)$ can also be made up of several objectives, some of which may be conflicting.
(iv) The feasibility set, S, may not be necessarily convex and may even include disconnected subsets.
(v) And finally, the viability of the parameter values within the definition of F and S can be critical as these may be probabilistic, estimated or even not fully known.

In brief, when set S is a discrete set, the problem (P) falls into the category of discrete optimisation problems (also known as combinatorial optimisation), whereas if it is a continuous set, (P) is referred to as a continuous optimisation problem (also known as global optimisation).

Local versus Global Optimality

Consider $X \in S$ and let $N(X) \subset S$ be a given neighbourhood of X. $N(X)$ could be defined by a 'small' area around X. Some definitions of the neighbourhoods will be defined later depending on the application.

Local minimum (maximum)

$\widetilde{X}$ is a local minimum (maximum) with respect to neighbourhood $N(.)$ if $F\left(\widetilde{X}\right) \leq (\geq) F(X) \forall X \in N(X)$

Global minimum (maximum)

X^* is a global minimum (maximum) if $F\left(X^*\right) \leq (\geq) F(X) \forall X \in S$

For instance, if Ω represents all the possible neighbourhoods and Λ the set of all local minima (maxima), X^* can also be defined as $X^* = \text{Arg M}$ in $\{F(X); X \in \Omega(X)\}$ or $X^* = \text{Arg Min}\left\{F\left(\widetilde{X}\right); \widetilde{X} \in \Lambda\right\}$.

In brief, the global minimum (maximum) X^* is the local minimum (maximum) that yields the best solution value of the objective function F with respect to all neighbourhoods.

Local Search

This is the mechanism (i.e., operator or transformation) by which $\widetilde{X}$ is obtained from X in a given neighbourhood $N(X)$. In other words, $\widetilde{X} = \text{ArgMin}\{\text{F(X)}; \text{X} \in \text{N(X)}\}$.

1.3 Possible Methodological Approaches

Despite the above difficulties, many applications fall into suitable formulations where optimality can be guaranteed (e.g., optimal solutions can be determined). In this book, we concentrate on those cases where an

optimal solution cannot be guaranteed while a 'good' solution, which may or may not be optimal, is required for practical purposes.

Most general optimisation methods fall into two main categories known as exact (optimal) algorithms and approximate or heuristic algorithms. The former is capable, at least theoretically, to guarantee the optimal solution in a finite number of steps, whereas the latter category includes heuristics. These are based on a set of rules which can be developed from experience, problem characteristics, common sense and mathematical logic. They have the tendency to tackle the problem on its entirety and in a reasonable amount of computer time. The solutions produced by this class of methods are, unfortunately, not necessarily optimal. However, the performance of these methods can be evaluated using certain performance criteria which will be briefly highlighted in subsequent sections of this chapter.

It is also worth noting that it may be useful to distinguish between heuristic and approximation algorithms as both aim to yield a feasible solution. The only difference is that the approximation methods have a certain guarantee of the solution quality based on the worst case scenario. For instance, for the travelling salesman problem (TSP), the well-known Christofides' algorithm which is based on solving the minimum spanning tree has a worst quality ratio of 1.5 (cost of the approximate TSP tour/cost of the optimal TSP ≤ 1.5). Also, a simpler approximate method in bin packing is the next fit (NF) algorithm where the ratio is 2. This information, though theoretically interesting, needs to be carefully translated to the user as it could unfortunately be wrongly understood by the user. Most heuristics on the other hand, have no similar mathematical bounds for quality purposes though some special ones may have, and some researchers are currently attempting to find them. There are other useful ways for evaluating the quality of a given non-optimal approach including heuristics which I shall revisit later in this chapter.

When approaching complex real-life problems, I consider the following four commonly applied steps to be followed in the sequence given below. Note that this ordered list is not exhaustive as other combinations do exist though these may be relatively more difficult to define explicitly.

The aim is to apply

1. an exact method to the exact (true) problem; if not possible go to (2).
2. a heuristic method to the exact problem; if not possible go to (3).
3. an exact method to a modified (approximated) problem; if not possible go to (4).
4. a heuristic (or approximate) method to the approximated problem.

Though these rules are presented in the above ranking, the complexity in the design of the heuristic in Step (2) which aims to retain the true characteristics of the problem, and the degree of modification of the problem in Step (3) are both crucial points when dealing with practical problems. It may be argued that Steps (2) and (3) could swap places. The idea is to keep the characteristics of the problem as close as possible to the true problem (small simplification only) and then try to implement (1) or (2). The following recursive process, as given in Algorithm 1.1, can be adopted when approaching complex real-life problems.

Algorithm 1.1: A Possible Recursive Methodological Approach

Step 1 Define (P) as the current problem.
Step 2 Choose a suitable exact method to solve (P).
If it is possible, then conduct the experiment and stop; otherwise go to Step 3.
Step 3 Choose a suitable heuristic to solve (P).
If it is possible, then conduct the experiment and stop, else modify (P) and go back to Step 1.

The level of modification ought to be carefully considered. A massive modification may make the problem very simple to solve, but will have little resemblance to the original problem, whereas a tiny modification may lead to a problem that could still be hard to solve.

Another related approach would be to start with a simplified or a relaxed version of (P), find a solution and check whether those complex constraints are satisfied. If it is the case, there is no need to worry about the original problem as the solution is optimal. If some constraints are

violated, which is likely to happen especially at the beginning of the search, additional characteristics are then introduced gradually. The process is repeated until the problem becomes impractical to solve, and hence the feasible solution found at the previous stage can then be used as the final solution of the problem. Note that this concept is the basis for dual-based methods such as the dual simplex method in linear programming, and also in Lagrangean Relaxation, a topic I shall visit in the hybridisation chapter.

In summary, we can say that the modifications can be implemented at three levels, namely, the input stage (problem characteristics), the algorithm stage (whether it is an exact or a heuristic method) and, finally, the output stage (solutions found). The choice of whether to concentrate on the input, the algorithm or the output is critical. I think one way forward is to try to retain the input to its originality as much as possible and to focus on the other two. Obviously if slight changes made to the input lead to a problem that can be solved optimally or efficiently, such an approach should not to be discarded.

The hierarchy presented in Algorithm 1.1 is used to emphasise the need to maintain the problem characteristics as close as possible to the ones of the *true* problem. This concept is highlighted by the following observation that *it is better to have a good and acceptable solution to a true problem rather than an optimal solution to a problem that has very little resemblance to the original problem (e.g., solving the wrong problem)*.

The lack of understanding, and in some cases, the lack of appreciation of the difficulties of the real-life problem, could lead to a less favourable relationship between the two main parties, namely, the end user (company) and the consultant or researcher. This unfortunate scenario may encourage practitioners to distant themselves from academics, which, in turn, may lead to the least favourable outcome for either party. This trend is becoming less and less strong as nowadays the relationship between universities and the outside world steadily improves as mutual benefits are now being generated and measured. The company gains an added competitive advantage, while the academic enhances his/her research portfolio. Furthermore, for example, in the UK, the research impact in practice is now used towards the overall research assessment score for the school or

faculty involved as required by the Research Excellence Framework; see REF (2014).

1.4 Need for Heuristics

I would like to stress once more that heuristics are used only when exact methods, which guarantee optimal solutions, are impractical because

(a) either the computational effort required is excessive, or
(b) the risk of being trapped at a local optimum is too high.

Given the above aspects, in such circumstances, heuristics become virtually the only way to help the user find reasonably acceptable solutions. The reasons for accepting and promoting heuristics include the following with some being noted in Salhi (1998, 2006).

1. Heuristics can be the only way forward to produce concrete solutions to large and difficult combinatorial and global optimisation problems.
2. They are easily adaptable and accessible for additional tasks or constraints if necessary.
3. Heuristics can be easily supported by graphical interface which can help the user in understanding better the progress of the search, and in modifying the outcomes if need be, hence avoiding the drawbacks of black box products.
4. Management and less specialised users find them reasonably easy to understand and therefore to interact with.
5. These methods are not difficult to code, validate and implement.
6. Management can introduce some unquantifiable measures indirectly to see their effect as solutions can be generated while conducting an interesting scenario analysis.
7. These methods are suitable for producing several solutions and not only a single one, thus providing the user with the confidence, flexibility and the opportunity to choose one or more solutions for further investigation if need be.

8. The design of heuristics can be considered as an *Art* since a proper insight of a problem is fully required, but also as a *Science* as it requires some form of logic. This interesting combination can bring the user and the researcher closer together at the design stage, enabling both parties to contribute positively to the success of the project.

1.5 Some Characteristics of Heuristics

The last item fits well within the philosophy of science as presented by Popper (1959) who emphasised that the design of heuristics ought to favour insight over efficiency, and also that learning is part of a process of exploiting the failures of heuristics.

The following characteristics are worth considering in the design of a given heuristic though these do not have to be followed exactly. Some are by-products of the attributes that make heuristic necessary as mentioned in the previous section, whereas some are added for generalisation purposes. For simplicity, these characteristics are only summarised below while noting that more details can be found in Laporte et al. (2000), Salhi (1998, 2006) Hansen et al. (2010) among others:

(i) Simple and coherent—the method should follow well-defined steps.
(ii) Effective and robust—it should be reliable enough to provide good or near optimal solutions irrespective of the instances used.
(iii) Efficient—the time required needs to be acceptable taking into account whether the problem is strategic or tactical. Either way the heuristic ought to be implemented in an efficient way as will be highlighted in the implementation chapter.
(iv) Flexible—this needs to cater for modification and adaption. For instance, the addition/deletion or modification of some steps can be easily accommodated to include new ideas and to be able to solve related problems while retaining the other strengths of the heuristic. This obviously includes the flexibility in allowing interaction with the user as he/she is the one who takes control and has the final decision.

Heuristics, in general, including global search heuristics (also called modern heuristics, metaheuristics or meta strategies) that shall be presented throughout this book have the properties to exploit the overall region rather than individual or local regions. However, two natural questions arise:

(a) Is there any guarantee that escaping from a local optimum will turn into exploring new and more promising regions of the feasible set?
(b) Do such regions exist or do we expect the local minima (maxima) to be uncorrelated?

The answers of these questions are far from being definite but provide a platform to base the search when designing a heuristic. Theoretically the answer to (a) is no, but the principle of getting closer to a yes could be helpful when constructing diversification strategies. For (b) the local minima are theoretically uncorrelated in general though some local optima may not be far from each other.

Metaheuristics

Note that metaheuristics (also known as modern heuristics) are considered as higher level heuristics that are devised to guide other constructive heuristics or local searches to reduce the risk of being trapped into a poor local optimum. Though this definition helps in differentiating between low-level heuristics, which could include, for example, constructive and simple heuristics as well as local searches, and higher level ones that aim to control the lower ones, in this monograph I shall refer to the word heuristics whenever possible irrespective of their level. The aim is to keep the terminology simple while providing freedom and flexibility to the word heuristics as it was historically described. In other words, it relates to mechanisms used to discover and search, using ways that may be simple (low-level heuristics or local search) or more complex and powerful (metaheuristics and others to mention later in this book).

1.6 Complexity and Performance of Heuristics

Heuristics need not be linked to the phrase usually known as 'quick and dirty' or 'guess work'. Heuristics are wider in concept than that those simple phrases though in some situations simple implementations may do the job.

Heuristics ought to be carefully devised to represent the full characteristics of the problem (not necessarily the generalised problem), and validated and tested from both points of view, namely, computing time and solution quality. The main criteria for evaluating the performance of a new heuristic may be classified under two headings, namely, the quality of the solutions provided and the computational effort, measured in terms of computer processing units (CPU) time. Other criteria such as simplicity, flexibility, ease of control, interaction and friendliness can also be of interest, and more particularly to the user (see Section 1.5). For further details, see Reeves (1995), Salhi (1998, 2006), Barr et al. (1996), Johnson (1996) and Laporte et al. (2000).

Solution Quality

There are at least five performance measures required to test a given heuristic. These include empirical testing, worst case behaviour, probabilistic analysis, lower bound (LB) solutions and obviously benchmarking.

(a) *Empirical testing*- this can be based on the best solutions of some of the existing heuristics when tested on a set of published data. Here, we can produce average deviation, worst deviation, the number of best solutions and so on. This measure, which is one of the most useful and commonly used approaches in practice, is simple to apply and can be effective when published results exist. Although accuracy is guaranteed as the results obtained are known with certainty, this type of analysis can provide only statistical evidence about the performance of the heuristic for other not yet tested instances.
(b) *Worst case analysis*—a pathological example, which is represented purposely to show the weakness of the algorithm, needs to be

constructed. It is usually very hard to find such an example especially if the problem is complex. One of the drawbacks of such an analysis, though theoretically strong, is that in practice the problem under study rarely resembles the worst case example. It is however reassuring that at least such a heuristic produces in the worst scenario solutions that are α % inferior at most. However, this information can also be misleading, and could put the user off if such an information is not communicated appropriately. One way is to understand the problem rather well and then construct a worst case example for a class of problems sharing similar characteristics to the real problem. Such a result provides a useful measure and a guaranteed performance which is not too far away from the real story. This measure is similar to the quality bounds usually developed for approximation algorithms (first fit decreasing [FFD] for bin packing, Christofides spanning tree based for the TSP, etc.).

(c) *Probabilistic analysis*—the density function of the problem data needs to be determined, allowing statistical measures to be derived such as average and worst behaviour.

(d) *Lower bounds*—one way is to solve a relaxed problem where many of the difficult constraints are removed (e.g., LP relaxation), or where the transformed problem falls into a nice class of easy problems (Lagrangean relaxation). The main difficulty is that the LB solutions obtained have to be rather tight to tell the quality of the heuristic solution; otherwise, misleading conclusions could be drawn.

(e) *B*enchmarking—in situations where a benchmark solution already exists, which is given by the user, the obvious way is to see how the solution found by the new heuristic compares with the benchmark. This can have positive impact in both the design of the heuristic (as initial results may not be necessarily competitive and hence enhancements could be added making the heuristic more efficient and powerful) and the way the user conceives the result. This is a win-win situation because if the results are better, the company could have a competitive advantage over its competitors while learning where improvements can be made. But, if the results happen to be the same, this could also demonstrate that what the company is doing

is good as it is now formally tested by an outsider. This may lead to a growth in confidence and self-belief for the user besides gaining additional trust from senior management.

The relationship between the user and the analyst is of crucial importance at this level especially if the user is aware of the primary runs of the heuristic as discussed in the earlier point (e). Inferior or infeasible solutions can send the wrong signal to the user, and therefore, a good understanding on the progress of development of the heuristic is vital. This extra human effort does not only avoid communication hick-ups from escalating, but also helps in building a friendly atmosphere in which modification and improvement are part of the design that needs to take place. In some cases, the initial runs are not shown to the user, and only when positive results are found will the user commence to inject feedback. The former strategy can be more effective if it is well looked after, whereas the latter one is good as long as the user is not supposed to react negatively in early stages of the work. A strategy that sits in between the above two ideas may be the most appropriate in many cases; however, the breaking point is dependent on the analyst's relationship with the user, the company's reputation, the routine checks which are initially arranged between the user and the analyst and the urgency by which the end product needs to be delivered.

Computational Effort

Computational effort is usually measured by the time complexity and the space complexity of a solution procedure. The former describes the computing time the method requires for a given instance, whereas the latter measures the storage capacity needed when solving a given instance. Unfortunately, the latter is seldom discussed in the literature.

Time Complexity

The time complexity of an algorithm is defined by $O(g(n))$ where n denotes the size of the problem.

If $g(n)$ is a polynomial function of n (e.g., $an^k, k \geq 1, a > 0$), then the problem can be solved within a reasonable amount of computation time. But if $g(n)$ is an exponential function of n, the problem may be difficult to solve. Such types of problems are usually known to be *NP* hard, see Garey and Johnson (1979) for more details.

For instance, in bin backing, the aim is to minimise the number of bins of equal volume or weight to be used where the weight of each of the n items to be stored in the bins is known. The FFD heuristic has a runtime $O(n^2)$, whereas the NF heuristic is relatively quicker having a runtime $O(n)$. However, FFD has a tighter worst quality ratio of $\frac{N_{FFD}}{N^*} < \frac{11}{9} + \frac{4}{N^*}$, whereas $\frac{N_{NF}}{N^*} \leq 2$, with N^* representing the minimal number of bins and N_H the number of bins found by heuristic H.

Space Complexity

Although this issue is less referenced when compared to time complexity, the way the data are stored and retrieved is an important issue in heuristic design. An efficient data handling not only uses the smallest necessary storage capacity in the computer but it can also save a large amount of computing time by not calculating unnecessary information which is either redundant or already found in earlier iterations. The issue of memory excess is also well known when using commercial optimisation software such as CPLEX, ILOG, LINDO, Xpress-MP, GuRobi and so on. The problem may not be resolved because of the computing time required, but simply because the computer runs out of memory. A massive storage space may be needed for the software to sometimes even start the optimisation or during the search due to the large amount of branching within Branch and Bound (B&B). However, there can sometimes be ways around the problem. For instance, in some cases, using another branching strategy within CPLEX such as depth first instead of best first helps in reducing the storage burden up to a certain point.

'Real' Meaning of Large or Small Computing Time

It is worth highlighting that the concept of large or small computer time ought to be relative to both the nature of the problem (strategic, tactical or operational) and the availability of the computing resources. The time for interfaces is usually ignored in research though it can constitute an important part of the total computing time in practice. This additional time could however be taken to be a constant if carried out by professionals in software engineering or related areas.

The impact of computing effort is directly related to the importance of the problem. For instance, if the problem needs to be solved once or twice a day (very short-term planning), it is essential that the algorithm be quick, whereas if the problem is solved at a medium or at a strategic level (once every month or year), it becomes less important to give a high priority to the CPU time, but more so to the quality of the solution. Consider the case where the problem requires a large investment such as with the location of new facilities, the purchase of expensive equipment, planning of the workforce, among others. In these instances, it does not matter so much if the method takes ten minutes, five hours or even a day, as long as good solution is provided at the end. However, one may also argue that the algorithm should not be too slow for simulation purposes even when tackling strategic problems, so several options and scenarios could be investigated providing the decision makers with additional flexibility. For instance, it may be that the third best solution, when taking other factors into account (i.e., unquantifiable or even confidential aspects) besides the cost, becomes the most practical option to choose from all available solutions.

A good and efficient computer code can obviously save a lot of unnecessary computing time. This can be achieved by avoiding redundant calculations of already computed full or partial information. A good data structure (DS) which keeps track of already computed information could also help. The reduction in computing time can also be obtained by introducing some reduction tests (neighbourhood reduction) which eliminate testing certain cases (or combinations) which in the analyst's view are unlikely to influence the final best solution. There will generally be a trade-off between speed by which the best solution is obtained and the

quality of that solution. I shall visit this important aspect when discussing the implementation issues in Chap. 6.

1.7 A Possible Heuristic Classification

There are several ways to classify heuristics including the following:

(i) Deterministic vs stochastic
(ii) One solution at a time vs several solutions taken simultaneously (population)
(iii) Fast and dirty vs slow and powerful
(iv) Classical heuristics vs modern heuristics

In this monograph, the following categorisation consisting of four groups that are not necessarily entirely disjoint, as may be conceived by some researchers, is provided:

(i) Improving solutions only
(ii) Not necessarily improving solutions
(iii) Population-based
(iv) Hybridisation

The first two groups are completely disjoint, whereas the last two could interrelate with each other as well as with the first two. For clarity of presentation, such links are highlighted in Fig. 1.1 in dash, whereas the main links are shown in bold.

1.8 Summary

A general view of heuristics is given followed by the need for their usage in practice. As these methods are not exact, some performance measures and appropriate characteristics are provided to guide the user when designing such techniques. A simple classification of heuristics is also provided that

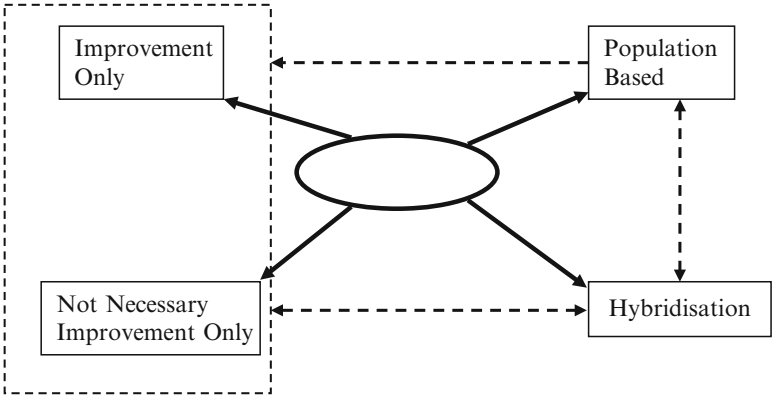

Fig. 1.1 A heuristic classification

will be used throughout this monograph. In the next chapter, I shall concentrate on the first category, namely, improvement-only heuristics.

References

Barr, W. S., Golden, B. L., Kelly, J. P., Resende, M. G. C., & Stewart, W. R. (1996). Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics, 1*, 9–32.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. San Francisco: Freeman.

Hansen, P., Mladenović, N., Brimberg, J., & Moreno Perez, J. A. (2010). Variable neighbourhood search. In M. Gendreau & J. Y. Potvin (Eds.), *Handbook of metaheuristics* (pp. 61–86). London: Springer.

Johnson, D. S. (1996). A theoretical guide to the experimental analysis. *AT&T Research Report, Bell Laboratories, University of Michigan*.

Laporte, G., Gendreau, M., Potvin, J.-Y., & Semet, F. (2000). Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research, 7*, 285–300.

Popper, K. (1959). *The logic of scientific discovery*. London: Hutchinson.

Reeves, C. R. (1995). *Modern heuristic techniques for combinatorial problems*. Oxford: Blackwell Scientific Publications.

REF. (2014). REF 2014: Research Excellence Framework. http://www.ref.ac.uk

Salhi, S. (1998). Heuristic search methods. In G. A. Marcoulides (Ed.), *Modern methods for business research* (pp. 147–175). New Jersey: Lawrence Erlbaum Associates.

Salhi, S. (2006). Heuristic search in action: The science of tomorrow. In S. Salhi (Ed.), *OR48 Keynote papers* (pp. 39–58). Birmingham, UK: ORS Bath, The OR Society.

2

Improvement-Only Heuristics

2.1 Neighbourhood Definition and Examples

I first present the basic definition of neighbourhood followed by some simple combinatorial type examples that are commonly used in the Operational Research/Computer Science literature.

Neighbourhood Definition A solution, say $X \in S$, has an associated set of neighbours, say $N(X) \subset S$, which is called the neighbourhood of X. A solution, $X' \in N(X)$, can be obtained directly from X by an operation which is referred to as a move (or transition m), say $X' = m(X)$. Such a move can be a single move consisting of simple or complex operations, or a series of moves put together that can be used either in a well-defined and deterministic sequential manner, or in a random or a pseudo-random way.

The Travelling Salesman Problem

For instance, for the case of the Travelling Salesman Problem (TSP) where the aim is to get the least cost tour that originates say from a

S. Salhi, *Heuristic Search*, DOI 10.1007/978-3-319-49355-8_2

home town (a depot or a facility), visits each city only once and returns back to the home town. The idea is to have a sequence of nodes (cities, say $i = 0, 1, \ldots, n$) starting from the home town say 0. Let d_{ij} denote the distance between city i and city j where $i, j = 0, 1, \ldots, n$.. Let $\sigma = (0, \sigma_1, \ldots..\sigma_n, 0)$ be the sequence where σ_j denotes the city occupying the j^{th} position in the tour and $F(\sigma)$ the total travel distance (cost) of the tour $\sigma \in \Omega$ where Ω represents all the possible sequences. The objective is to find the sequence with the least total distance (i.e., find $\sigma^* = \text{ArgMin}\{F(\sigma), \sigma \in \Omega\}$).

One possible neighbourhood for the TSP would be to exchange the position of a city with one of its neighbours. For example, the position of the i^{th} and the j^{th} cities in σ could be swapped leading to σ' as follows:

$$\sigma = (0, \sigma_1, \ldots, \sigma_i, \ldots, \sigma_j, \ldots, \sigma_n, 0) \rightarrow \sigma' = (0, \sigma_1, \ldots, \sigma_j, \ldots, \sigma_i, \ldots, \sigma_n, 0).$$

The *p*-Median Problem

Suppose we have n customers that need to be served from p facilities. The aim is to find the location of these p facilities out of m potential sites such that the average distance (or the total distance) from these p open facilities to all customers is minimised. Let the facility configuration of these p facilities be $\sigma = (\sigma_1, \ldots, \sigma_p)$ where $\sigma_j \in \{1, \ldots, m\}$ represents the j^{th} chosen facility and Ω being the set of possible configurations. The objective is to choose the configuration $\sigma^* = \text{ArgMin}\{F(\sigma), \sigma \in \Omega\}$ where

$$F(\sigma) = \sum_{i=1}^{p} \sum_{j \in A_i} d_{j\sigma_i} \text{ with } A_i = \left\{ j \in \{1, \ldots, n\} \mid d_{j\sigma_i} = \text{Min}\left(d_{j\sigma_k}; \sigma_k \in \sigma\right) \right\}$$

For the p-median problem, a possible neighbourhood would be to close an open facility say $\sigma_i \in \sigma$ and choose a potential site from the non-selected ones, say $k \notin \sigma$. In brief, we moved from σ to its neighbour σ' as follows

$$\sigma = (\sigma_1, \ldots, \sigma_i, \ldots, \sigma_p) \rightarrow \sigma' = (\sigma_1, \ldots, k, \ldots, \sigma_p).$$

In other words, $N(\sigma) = N(\sigma) - \sigma_i \cup k$
where $\sigma_i \in \sigma$ and $k \notin \sigma$.

The Bin Packing Problem

Consider there are n items of weight $w_j; j = 1, \ldots, n$ and bins of equal weight (can be different as well) W. The aim is to minimise the number of bins such that all the items are included. Assume that we have already obtained p sequences (bins) found by one of those approximation algorithms such as the FFD. We consider $p > LB$ with the lower bound LB taking the smallest integer that is larger than $\frac{\sum_{j=1}^{n} w_j}{W}$, otherwise the solution is obviously optimal. Let $B_k = \left\{B_k^1, ..., B_k^j, ..., B_k^{N_k}\right\}; k = 1, \ldots, p$ be the k^{th} bin with B_k^j representing the j^{th} item in bin k and N_k being the number of items in bin k. Let $W_k = \sum_{j=1}^{N_k} w_{B_k^j}$ be the total load in bin k. If infeasibility is not allowed, one possible move is to shift one item from one bin to another as long as the total weight of the receiving bin is not violated. For instance, remove item B_k^j from bin k and inserting it in bin l if $W_l + w_{B_k^j} \leq W$. This process is repeated until one bin is emptied.

The Vehicle Routing Problem

When there is a restriction on either the total travel time or/and the maximum total load to be transported on a vehicle (truck), the TSP may no longer be valid as the tour may use a larger load than the vehicle capacity say Q and could take longer than the daily travel time, say T. Consider there are n customers that need to be served from the depot which we refer to by 0. Each customer has its own demand say

$q_j; j = 1, \ldots, n$. Each vehicle has the same capacity Q and could not travel more than T (say nine hours per day including break). Here, all vehicle routes start and finish at the same depot. The aim is to find the number of vehicles used with their respective sequences (vehicle routes) so that the total distance travelled is minimised. Note that when Q and T are both very large, the problem reduces to the TSP with the depot as the home city.

Consider we have p routes and $R_k = \left(0, R^1_{k,}, \ldots, R^i_k, \ldots, R^j_k, \ldots, R^{N_k}_k, 0\right)$; $k = 1, \ldots, p$, with R^j_k denoting the j^{th} customer in the k^{th} route and N_k being the number of customers in route k. Possible neighbourhoods based on R_k include

(i) swapping the places of the i^{th} and the j^{th} customers (say R^i_k with R^j_k) in R_k to obtain say $R'_k = \left(0, R^1_{k,}, \ldots, R^j_k, \ldots, R^i_k, \ldots, R^{N_k}_k, 0\right)$,

(ii) moving customers between routes say customer R^i_k from R_k is moved to R_l and customer R^i_l from R_l to R_k making the two new routes $R^{'}_k$ and $R^{'}_l$ $R^{'}_k = \left(0, R^1_{k,}, \ldots, R^i_l, \ldots, R^j_k, \ldots, R^{N_k}_k, 0\right)$ and $R'_l = \left(0, R^1_{l,}, \ldots, R^i_k, \ldots, R^j_l, \ldots, R^{N_l}_l, 0\right); l \neq k$ as long as capacity and time constraints are not violated. The exchange does not need to be using the same customer position as shown here.

Non-linear Optimisation Problems

The classical non-linear optimisation problem (case of minimisation) can be formulated as

$$(P): \begin{cases} \text{Minimise} & F(X) \\ st & X \in S, S \subseteq R^n; X = (x_i),\ x_i \in [L_i, U_i], i = 1, .., n \end{cases}$$

The neighbourhood can be expressed as $N(X) = \left\{X' \in S \quad \text{such that } d(X, X') \leq R\right\}$ where R is the radius of the neighbourhood defined either as threshold or by upper and lower bounds of the coordinates $x_j; j = 1, \ldots, n$. One way would be to take x_j from the incumbent

solution $X = (x_i)_{i=1,\ldots,n}$ and replacing it by its neighbour $x_j^{'} \in [L_j, U_j]$ to obtain X' as follows:

$$x_j^{'} = \begin{cases} L_j & \text{if} \quad x_j - \alpha(U_j - L_j) \le L_j \\ U_j & \text{if} \quad x_j + \alpha(U_j - L_j) \ge U_j \\ x_j - \alpha(U_j - L_j) & \text{otherwise} \end{cases}$$

In other words, $X = (x_1, \ldots., x_j, \ldots, x_n) \rightarrow X' = \left(x_1, \ldots, x_j^{'}, \ldots., x_n\right)$ with $\alpha \in [0, 1]$ randomly generated.

If the problem is further constrained, feasibility can then be imposed while generating $x_i^{'}$ or restricting it marginally while attaching some penalties to the violation as part of an augmented objective function. This is used as part of the Lagrangean relaxation which will be discussed in Chap. 5 and also embedding in other heuristics so their search could explore a wider search space.

2.2 Basic Descent or Hill Climbing Method

The main steps of the basic descent method (BDM) are summarised in Algorithm 2.1. BDM has two main components namely the generation of the initial solution and the way such a solution is improved via the selected neighbourhood $N(.)$. An efficient implementation and a good design of these two components are worth the effort as these will contribute to the final quality of the solution.

Algorithm 2.1: The Basic Descent Method

Step 1 *(Initial Phase)* Select an initial solution, say $X \in S$.

Step 2 *(Improvement Phase)* Choose a solution $X' \in N(X)$ such that $F(X') < F(X)$.

If there is no such X', X is considered as a local minimum and the method stops, else set $X = X'$ and repeat Step 2.