11

An Overview of Heuristics and Metaheuristics

Saïd Salhi and Jonathan Thompson

11.1 Introduction

We live in a world of limited resources, including food, water and energy. As society becomes more advanced, the need to make efficient use of these limited resources becomes more pressing. The study of optimisation enables many real-life problems to be solved and leads to many positive outcomes. For example scheduling the shift patterns of nurses can lead to optimal use of resources and shift patterns that are conducive to good staff wellbeing. The optimal routing of delivery trucks can lead to reduced carbon dioxide emissions. Optimal scheduling of jobs leads to efficient use of resources and increased profits for manufacturers.

There are many real-life applications that can be solved optimally by one of the exact optimisation techniques known to the Operational Research community such as linear programming, integer programming, non-linear programming and dynamic programming among others. However, there are

S. Salhi (✉)
Kent Business School, University of Kent, Canterbury, UK
e-mail: s.salhi@kent.ac.uk

J. Thompson
School of Mathematics, Cardiff University, Cardiff, UK
e-mail: thompsonjm1@cardiff.ac.uk

S. Salhi and J. Boylan (eds.), *The Palgrave Handbook of Operations Research*,
https://doi.org/10.1007/978-3-030-96935-6_11

many applications where the combinatorial effect of the problem makes the determination of the optimal solution intractable, and hence, these standard techniques become unsuitable. This is because the computer time needed to find the optimal solution becomes too large to be practical for real-life situations, or specialist software is required that is not easily usable by many companies and organisations. To overcome this drawback, heuristic methods were introduced with the aim of providing the user with reasonably good solutions in a reasonable time. Heuristics can be likened to algorithms which are step-by-step procedures for producing solutions to a class of problems. In many practical situations they seem to be the only way to produce concrete results in a reasonable time. To date, heuristic search methods have been widely used in a large number of different areas including business, economics, sports, statistics, medicine and engineering.

The word 'heuristic' originates from the Greek word 'heuriskein' that means discover and explore. Heuristics are also sometimes referred to as rules of thumb. They have been used throughout history as humans are natural problem solvers and can naturally construct heuristic methods. An example given by Sorenson [101] is that when deciding the trajectory of a spear to throw at a wild animal, it is more important that the trajectory is selected quickly rather than optimally. The time spent to calculate the optimal trajectory will almost certainly mean the creature has disappeared long before the spear is thrown. Therefore a quick but not precise trajectory is superior. However the scientific study of heuristics began more recently and still continues to this day, with numerous academic papers each year proposing new heuristic methods for a wide variety of problems. Figure 11.1 shows the number of publications listed in Scopus with 'heuristic' as a keyword over the last 10 years and shows an increasing trend throughout the decade showing that advancements in computer software and hardware have not removed the need for heuristic solution methods.

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. A good insight into the problem that needs to be addressed is essential. Slight changes to the problem description can lead to major changes to the model being applied, for example a slight change to a problem that was easy to solve exactly may change the complexity of the problem and necessitate the use of a heuristic.

In this chapter we will provide a short introduction to optimisation and state the need for heuristics. Heuristics can be classified into various categories and we will consider improvement-only heuristics in Sect. 11.2, heuristics that accept worsening moves in Sect. 11.3, and population-based heuristics

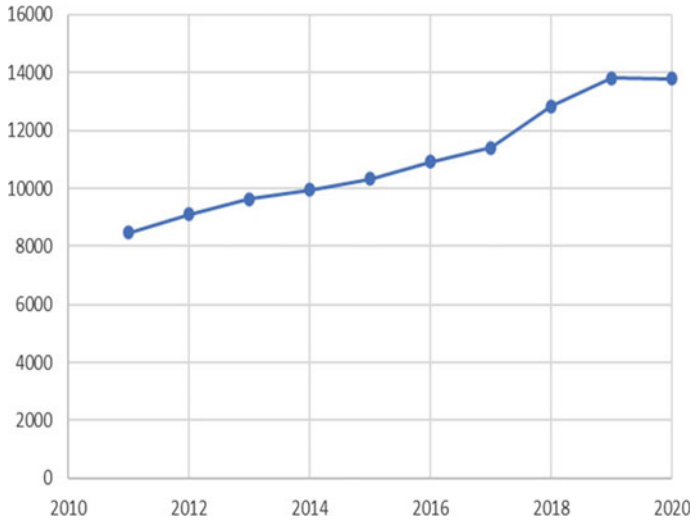

Fig. 11.1 The number of mentions of 'heuristics' on SCOPUS over the last 10 years

in Sect. 11.4. Some examples of applications will be provided in Sect. 11.5 and the chapter concludes with some suggestions for future research.

11.1.1 Optimisation Problems

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

$P =$ Minimise $F(X)$

s.t.

$X \in S$

Here, $F(X)$ defines the cost or value of each solution X. When the set S is discrete, (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).

11.1.1.1 Local vs Global Optimality

Consider $X \in S$ and let $N(X) \subset S$ be a given neighbourhood of X. $N(X)$ could be defined as being a set of solutions that are similar to X in some way. The decision of which neighbourhood to use in different applications is a vital one when applying an improvement heuristic.

$\widehat{X}$ is a **local minimum** (maximum) with respect to neighbourhood $N(X)$ if $F(\widehat{X}) \leq (\geq) F(X) \forall X \in N(X)$. X^* is a **global minimum** (maximum) if $F(X^*) \leq (\geq) F(X) \forall X \in S$. For instance, if Ω represents all the possible neighbourhoods and Λ the set of all local minima (maxima), X^* can also be defined as $X^* = \text{Arg Min}\{F(X); X \in \Omega(X)\}$ or $X^* = \text{Arg Min}\{F(\widehat{X}); \widehat{X} \in \Lambda\}$.

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 is the mechanism (i.e. operator or transformation) by which $\widehat{X}$ is obtained from X in a given neighbourhood $N(X)$. In other words, $\widehat{X} = \text{Arg Min}\{F(X); X \in N(X)\}$ denotes choosing the best member of the neighbourhood.

11.1.1.2 Modelling Approach

When approaching complex real-life problems, a possible modelling approach is to follow the following four steps. Note that this ordered list is not exhaustive as other possibilities do exist.

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 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 and then try to implement (1) or (2). Another related approach would be to start with a simplified or a relaxed version of (P), find a solution and check whether the solution satisfies the constraints to the original problem. 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. The choice of whether to use a heuristic method or to modify the problem into one that can be solved exactly is crucial. It appears that applying a heuristic is a straightforward approach and 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.

11.1.2 The Need for Heuristics

Heuristics are usually used only when exact methods, which guarantee optimal solutions, are impractical because the computational effort required is excessive. They are often used to produce solutions to problems in the class NP-complete, a set of problems for which there are no known efficient ways to find optimal solutions to large problems. However solutions to problems in this category can be verified efficiently. The most well-known example is the Travelling Salesman Problem (TSP) which consists of finding the shortest route to visit a set of cities and return to the starting point. The TSP is a computationally difficult problem and an optimal solution to even a relatively small problem can be difficult to find, leading to the widespread use of heuristics. The decision of whether to use exact or heuristic solution methods also depends on the context of the problem and how much time is available to find a solution. Heuristics may also be used where the problem is ill-defined, or a solution of reasonable quality is all that is required. Salhi [87], [92], [93] provides further reasons for accepting and using heuristics.

11.1.3 Some Characteristics of Heuristics

The following characteristics are worth considering in the design of a given heuristic. Some are by-products of the attributes that make heuristic necessary as mentioned in the previous section, whereas others are added for generalisation purposes. For simplicity these characteristics are only summarised below while more details can be found in [63], [87], [92], [93], [48] among others.

Heuristics should be:

i. Simple and coherent—the method should follow well defined steps which are not ambiguous in any way.

ii. Effective and robust—the method should be reliable enough to provide good or near optimal solutions irrespective of the instances solved.
iii. Efficient—the run time needs to be acceptable. The solution time will depend to some extent on how the method is implemented.
iv. Flexible—so they can be easily adjusted to solve variants of the problem with minimum changes while retaining the strengths of the heuristic. This may include the flexibility to allow interaction with the user so they can take some control over the algorithm.

11.1.4 Performance of Heuristics

One difficulty with heuristics is measuring the quality of their performance, given they are typically used on problems for which the exact solution is unknown. The main criteria for evaluating the performance of a heuristic are the quality of the solutions provided and the computational effort, measured in terms of CPU time, the number of iterations or the number of function evaluations. Other criteria such as simplicity, flexibility, ease of control, interaction and friendliness can also be of interest, but are more difficult to measure precisely.

Solution Quality There are various ways in which the solution quality of a heuristic may be measured. These include:

- Comparison with lower or upper bounds, which is most useful when bounds are tight
- Benchmarking against optimal results for smaller problems where the optimal solutions are known
- A theoretical understanding of worst-case behaviour
- Comparison with other heuristic techniques

Many papers merely use the latter method for evaluating solution quality and this is fraught with danger, particularly where the authors of a particular heuristic have also implemented the other heuristic methods for the comparison. One positive step is that an increasing number of authors are making their code and results available for evaluation by others.

11.1.4.1 Computational Effort

Heuristics are also evaluated in terms of their computational effort. Computational effort is usually measured by how long a heuristic takes, and how

much memory it uses. 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. It should be noted that the concept of large or small computer time should be relative to both the nature of the problem (whether it is strategic, tactical or operational) and the availability and quality 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. The importance of computing effort is directly related to the importance of the problem. So if the problem relates to short-term planning and needs to be solved several times a day, it is essential that the algorithm is quick, whereas if the problem is solved at a medium or at a strategic level (so once every month or year), the cpu time is less important, and more consideration can be given to the quality of the solution. So for example a supermarket routing delivery vehicles need a quicker solution than a university scheduling its examinations twice per year. It should be noted that for some strategic problems, users may require the need to investigate different options and scenarios in which case the heuristic would need to be run several times and therefore should not be too slow.

11.1.5 Heuristic Classification and Categorisation

There are several ways to classify heuristics. Heuristics can be greedy where solutions are constructed from scratch or improvement where an existing solution is improved over time. Other classifications take into account whether the heuristic is deterministic or stochastic, uses memory or is memory-less, and whether it considers one solution at a time or a population of solutions. Salhi [93] categorises heuristics into four groups, namely, (a) improving solutions only, (b) not necessarily improving solutions, (c) population-based and (d) hybridisation. The first two groups are completely disjoint, whereas the last two could interrelate with each other as well as with the first two. We discuss the first three of these in the next three sections. Hybridisation is discussed in the following chapter.

11.2 Improvement-Only Heuristics

In this section, we briefly discuss approaches that only allow improving moves. The simplest such method is a descent or hill-climbing method but more sophisticated improvement-only methods commonly used include

GRASP (Greedy Randomised Adaptive Search Procedure), multi-level, variable neighbourhood search, perturbation schemes, adaptive large neighbourhood search, iterated local search and guided local search. Some of these will be described below

11.2.1 Hill-Climbing Methods

The basic hill-climbing or descent method has two main components, the generation of an initial solution and the means of improving this solution. The initial solution is typically generated either randomly or using a simple constructive type heuristic that builds the solution piece by piece until the final solution is constructed. For example for the Travelling Salesperson Problem (TSP), the starting solution may be constructed by selecting the nearest unvisited city at each step. Different heuristics may be used instead, which may be more computationally expensive but could result in better quality starting solutions. For example for the TSP, the heuristic could insert an unvisited city into the partial tour in the position which causes the minimum increase in the tour distance. Typically these rules are myopic (short-sighted) so can lead to large increases in cost towards the end of the construction.

The initial solution is then subject to improvement. The process of improvement requires a cost function $F(s)$ which attributes a cost or value to each solution s, and a neighbourhood $N(s)$ that defines a set of solutions that can be reached from the current solution via a small change. For the TSP, the cost is the total length of the tour and the neighbourhood could be defined as the set of solutions constructed by deleting 2 edges from the current solution and then forming a new tour, or swapping the positions of two cities in the tour, etc. A selection mechanism also needs to be defined that determines which neighbouring solutions are sampled. The neighbourhood structures and the selection mechanism adopted are crucial elements that contribute considerably to the success of improving heuristics. The most commonly used selection strategies are (i) the best improving move where we evaluate all or a part of the neighbourhood and select the feasible move that yields the best cost, or (ii) the first improving move where we select the first feasible solution that is better than the current solution. Note that (i) may take longer as all moves have to be evaluated at each stage but do mean the heuristic ends in a locally optimal solution and the search can terminate as soon as none of the neighbouring solutions provides an improvement. In (ii) neighbouring solutions are sampled at random so the selection is much quicker and at first, it

is often straightforward to find improving moves. However as the number of iterations increases, fewer moves result in improvements and the search can only estimate that a locally optimal solution has been reached. Such heuristics normally terminate when there have been a certain number of iterations N_{end} without improvement, and the value of this parameter N_{end} depends on the size of the problem/neighbourhood and is problem and instance dependent. In this case, the search may need to be repeated several times as different solutions may be produced depending on the random number stream used.

Algorithm 1 summarises the main features of (a) the best improving move and (b) the first improving move.

Algorithm 1: The Basic Descent method

(a) Best Improvement

Select initial solution $s \in S$
Repeat the following steps:
Find the best member $s' \in N(s)$
If $f(s') < f(s)$ then
    Let $s = s'$
Else
    Terminate run

(b) First Improvement

Select initial solution $s \in S$
Set max_n = max number of consecutive moves without improvement
Set $n = 0$
Repeat the following steps while $n < max_n$
    Choose s', a random member of $N(s)$
    If $f(s') < f(s)$ then
        Let $s = s'$
        $n = 0$
    Else
        $n = n + 1$

It is also possible to employ a compromise strategy that sits between the best improvement and first improvement methods. This could be achieved in several ways, for example by selecting the best improving move found after a certain time has elapsed since the previous improvement.

These methods typically search a very small part of the solution space, and solution quality depends on the starting solution. Therefore they are of limited use but can provide quick improvements to solutions produced heuristically that may not be local optima.

11.2.2 Classical Multi-Start

The local search procedure is efficient when the objective function is unimodular (i.e. has one local minimum only). However, when there are several minima, it is impossible to get out of the neighbourhood of a local minimum

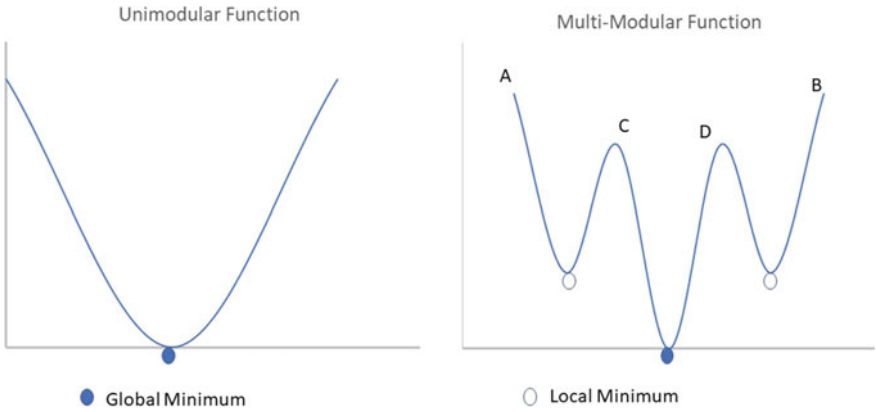

Fig. 11.2 Local optimality

using the same neighbourhood as shown in Fig. 11.2. In the second diagram representing a multi-modular function, if the search begins at position A or B, it converges to a poor quality local optimum whereas if it starts from position C or D, it converges to the global optimum. In practice for many combinatorial optimisation problems there are numerous local optima and the likelihood of finding a high-quality solution using a descent only strategy is quite small.

One way of improving the likelihood of finding a high-quality local optimum is to use a multi-start method where the search is repeated from different starting solutions to ensure a broader section of the solution space is searched. An example is Braysy et al. [8] who apply a multi-start local search method to the vehicle routing problem with time windows. They use a randomised constructive heuristic to generate a set of starting solutions and then apply local search to each. The best solution found is then subject to a further improvement process. In practice they construct 420 starting solutions but only apply the descent phase to solutions consisting of the minimum number of routes.

The basic multi-start method is summarised in Algorithm 2.

Algorithm 2: The multi-start method.

Step 1. Set $best_s = \infty$
Step 2. Repeat the following m times
(2a) Create an initial solution s
(2b) Perform a descent move to find a locally optimal solution s''
(2c) If $f(s'') < best_s$ then set $best_s = f(s'')$

If starting solutions are created randomly, then multi-start methods are well suited to parallelisation which can improve solution quality. However the search can be considered to be blind as it may re-visit local optima that have already been considered. Therefore methods that deliberately create starting solutions that differ to previously visited solutions are more popular. For example for the TSP, a list of the frequency of edges being included in solutions can be maintained, and future starting solutions can be weighted towards selecting edges that have not been frequently selected previously.

11.2.3 Greedy Randomised Adaptive Search Procedure (GRASP)

GRASP is an extension of multi-start methods and was introduced by Feo and Resende [39]. A more formal description is given in Feo and Resende [40]. It is a multi-start heuristic which consists of two phases, the construction phase that constructs an initial solution and a local search phase that improves the initial solution. This is repeated several times. GRASP can be considered as a memory-less multi-start heuristic. GRASP differs from multi-start in the way the initial solution is generated as it combines randomness with greediness. The idea is that at each step of the construction phase a selection rule is used that is less rigid and allows flexibility in choosing not necessarily the best option but to choose any options that are close in some way to the best. Typically a restricted candidate list (RCL) is used made up of either the top k best options or those options that are within a certain deviation from the best. For instance, for the case of minimisation: RCL= {set of attributes e such that $g_{\min} \leq g(e) \leq g_{\min} + \alpha(g_{\max} - g_{\min})$} where $0 \leq \alpha \leq 1$ and g_e is the cost of incorporating element e into the solution. The element to be added to the partial solution e^* is then randomly chosen from the RCL. Note that if $\alpha = 0$ this reduces to a greedy method whereas if $\alpha = 1$ the search becomes the classical random multi-start approach. The basic GRASP is described in Algorithm 3.

Algorithm 3: The GRASP method.

Step 1. Set $best_s = \infty$
Step 2. Repeat the following m times
(2a) Create an initial solution s using a greedy-randomised procedure and an RCL
(2b) Perform a descent move to find a locally optimal solution s''
(2c) If $f(s'') < best_s$ then set $best_s = f(s'')$

There are several possible variations to GRASP. For instance the choice of α does not need to be fixed beforehand but can be adjusted from one cycle to another depending on the solution quality found in previous solutions (this is known as reactive grasp). The definition of α can also be defined as a function of the gain/loss at a given iteration, or as a convex or a concave function dependent on the number of iterations. Path-relinking can be introduced which involves exploring the path between different locally optimal solutions to search for better solutions. Learning can also be introduced, either to guide the search towards new unexplored areas of the solution space or to maintain in future solutions attributes that appear to commonly occur in high-quality solutions. For examples see [46], [70] and [94].

11.2.4 Variable Neighbourhood Search (VNS)

Variable Neighbourhood Search (VNS) requires the definition of several neighbourhoods for the given problem. VNS attempts to escape local optima by moving to a new, usually larger, neighbourhood whenever there is no possible improvement through a local search in a given neighbourhood, and then reverts back to the first one, usually the smallest, if a better solution is found. VNS was originally developed by Mladenović and Hansen [75] for solving combinatorial and global optimisation problems. VNS contains mainly three phases, namely, a shaking phase (given a set of neighbourhood structures, a neighbouring solution is generated), an improvement phase (a local search or a local search engine is deployed to improve the currently perturbed solution) and finally a neighbourhood change phase (move or not move). VNS takes advantage of the fact that a local minimum with respect to a given neighbourhood may not be a local minimum for other neighbourhoods, but local optima for several different neighbourhoods are often close to each other. However, a global minimum is a local minimum for all neighbourhoods.

VNS starts from some starting solution x and local search is applied using an initial neighbourhood $N_1(x)$. The locally optimal solution found under the initial neighbourhood is then used as the starting solution for a local search under neighbourhood $N_2(x)$. If an improving move is identified, the search reverts to the first neighbourhood; otherwise the search moves onto neighbourhood $N_3(x)$ and so on. See Algorithm 4 for a description.

Algorithm 4: Variable Neighbourhhod Search (basic VNS).

Step 1.
(1a) Set $best_s = \infty$
(1b) Generate an initial solution s and define the neighbourhood structures $N_1, N_2, \ldots, N_{kmax}$
(1c) Set $k = 1$
Step 2. Repeat until stopping criteria are satisfied
(2a) Choose a solution s' in the neighbourhood $N_k(s)$ (shaking procedure)
(2b) Perform a descent move (a local search or a series of local searches) to find a locally optimal solution s''
(2c) If $f(s'') < best_s$ then set $best_s = f(s'')$
(2d) If $f(s'') < f(s)$ then Move or not
Set $s = s''$ and $k = 1$
Else
Set $k = k + 1$; If $k > kmax$, set $k = 1$

Compared to other powerful heuristics, VNS has the advantage of being simple and easy to implement as it requires only the definition of the neighbourhoods and the local search and there are no parameters to consider. A number of variants have been applied to different combinatorial problems with success. These include:

- Reduced VNS is useful for large problems where local search is too time-consuming and hence omitted from the search. In other words, this is mainly a VNS without the local search (i.e. the use of the shaking procedure and the move or not step) but allows to be used either in a systematic manner (reverting to the first neighbourhood, or to the next one, or a combination).
- VND, short for variable neighbourhood descent, is focused on the local search phase and the move or not move phase. Here the way the local search phase is implemented is critical. Usually there are a number of local searches used either in series or in a VNS type format like the multi-level heuristic described earlier.
- General VNS is a VNS where the local search phase is a VND.
- Skewed VNS explores areas of the solution space which can be far from the current solution by accepting non-improving solutions based on a certain threshold. In other words the move or not move phase is made slightly relaxed.

Note that in the improvement phase within the local search either a first improvement or a best improvement is adopted which can lead to several other variants. The move or not move step can also have several options such as sequential, cyclical or in a pruning way as defined in [49]. For more description on these and other variants, see [48], [49] and [93].

11.2.5 Iterated Local Search (ILS)

This consists of two main phases known as the construction/diversification phase and the local search phase. At each iteration the current solution is diversified, and then is subject to local search. These are performed repeatedly while an acceptance step is embedded to retain the best solution found so far. The aim is to ideally keep using the same local search as a black box while providing interesting solutions to it through diversification. This approach is simple and efficient and has shown to be powerful in obtaining interesting results in several combinatorial problems. One possible implementation of a strong ILS could be the combination of LNS (diversification/construction) with a VND or randomised VND (local search). For more details and possible applications of this heuristic, see [67].

11.2.6 A Multi-Level Composite Heuristic

Multi-level composite heuristics are similar to variable neighbourhood descent and involve improving the solution using a small number of local searches, known as refinement procedures in sequence, with the method stopping when there is no further improvement possible. The resulting solution should be a high-quality solution as it is a local minimum with respect to the selected procedures. The heuristic starts by finding a locally optimal solution using the first refinement procedure, and then this solution becomes a starting solution for a second refinement procedure, etc. In addition, once a new better solution is found, it can be used as an initial solution for any of the other refinements, not necessarily the next one in the list. For simplicity, we can restart the process using the first one which is the simplest refinement in our list of available refinements. This is the backbone of multi-level heuristics initially proposed in Salhi and Sari [90] for the multi-depot routing problem with a heterogeneous vehicle fleet. This approach has now been successfully applied to many other combinatorial and global optimisation problems. The choice of the refinement procedures, the sequence in which these are used and the choice of the refinement to go back to, once a new better solution is found, can be critical. Note that once a solution is found at a given level, it is appropriate to go to any of the other refinements of the earlier levels. A standard approach is to revert back to the first level where computations are relatively fast. More details on this approach can be found in [90], [96].

The way the levels are organised can have a significant effect on the solution quality and this requires careful consideration. The choice of successive

refinements may be critical as one may wish to consider the next local search to be drastically different in structure to the previous one so as to provide diversity. The search does not necessarily have to return to the initial refinement once a solution is determined and the choice of the refinement to return to does not need to remain constant throughout the run. It may be that the obtained solution would be better suited to another refinement chosen randomly or based on certain rules that consider the structure of the current solution. It can be a challenging task to match the structure of a given solution with the characteristics of the set of refinements used. The integration of learning within the search could render this approach more adaptive and hence more powerful as demonstrated in [103], [104].

11.2.7 Problem Perturbation Heuristics

The idea of problem perturbation heuristics is to perturb the original problem to obtain a series of gradually perturbed problems. Local search is applied to the initial problem and a locally optimal solution is then obtained. The problem is then perturbed in some way and local search is applied again to this new problem leading to a different solution. It can be considered whether this new solution is a better solution to the original problem than the previously found local optimum.

The perturbed problem may be found by adding a violation that means the current local optimum is infeasible. As there is a well-defined perturbation built up between successive perturbed problems, the successive solutions may have the tendency of retaining some of the important attributes in their respective configurations. After a set of perturbed problems are found, the search may proceed by gradually relaxing the problem by removing the violations again, one by one. This continues until the problem returns to the initial problem again. At this stage a local search to improve the solution can be used again and then either the process starts adding violations again or making the original problem even less restricted by removing some of the constraints. This process can be repeated several times.

For instance, consider the p-median problem where the objective is to identify the optimal location of a fixed number of facilities (say p) with the aim of minimising the total transportation cost. A local search can be performed and a local optimum identified. Then, the solution is allowed purposely to become infeasible in terms of the number of facilities by accepting solutions with more than or less than p (say $p \pm q$ where $q \in [p/4, p/3]$). By solving the modified problems, infeasible solutions can be

generated that when transformed into feasible ones may yield a cost improvement. Initially a feasible solution with p facilities is found, then one facility is added at a time and after each, a simple local search is activated. Once we reach $p+q$ facilities, we then start removing one facility at a time till we get to a feasible solution with p facilities where a more powerful local search is applied for intensification purposes to try to obtain a better local minimum. At this stage we also allow the solution to be infeasible by removing one facility at a time until we reach say $p-q$, from where we start adding facilities again until we reach a new feasible solution. This up and down trajectory makes up one full cycle. The process is then repeated until there is either no improvement after a certain number of cycles or the overall computing time is met. When this process is repeated several times going through several cycles, it acts as a filtering process where the most attractive depots will have the tendency to remain in the best configuration, which is a form of survival of the fittest. Such an approach was developed by Salhi [90] and it performed well when tested on a class of large facility location problems with known and unknown values of p. Zainuddin and Salhi [120] adapted this approach to solve the capacitated multisource Weber problem and Elshaikh et al. [38] modified it for the continuous p-centre problem.

A similar strategy is Strategic Oscillation where the search moves between the feasible and infeasible regions. Consider Fig. 11.3 which illustrates a solution space with a disconnected solution space. A search is performed in a

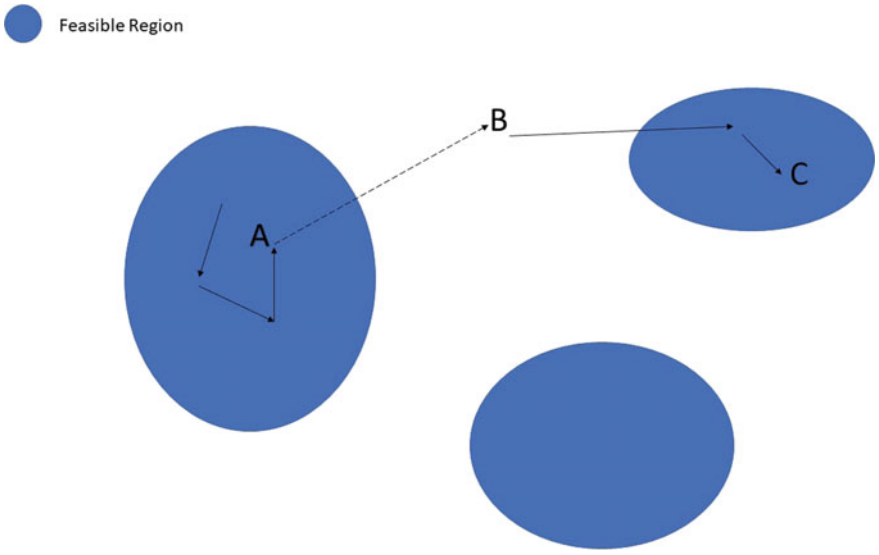

Fig. 11.3 Strategic oscillation to search a disconnected solution space

feasible region, ending in say point A but the constraints are then relaxed, enabling the search to move into the infeasible region (say point B). At this stage, the constraints are re-imposed and the local search is used to guide the search back towards a feasible region, point C in this case. In this way the search investigates a broader area of the solution space and is able to navigate between different parts of a disconnected solution space.

The amount of infeasibility allowed is dealt with by attaching penalties to broken constraints. These penalties are dynamically updated. This concept is useful particularly when the solution space is disconnected or non-convex as it is impossible to cross the infeasible region using only feasible solutions. The only difference with problem perturbation heuristics is that the latter is designed purposely to explore these infeasible solutions in a guided manner without any penalty attached so that some useful and promising attributes could then be identified.

11.2.8 Some Other Improving Only Methods

Large Neighbourhood Search (LNS): The idea is to use not necessarily small neighbourhoods but larger ones as well. The need for these large neighbourhoods is sometimes crucial to get out of local optimality as pointed out by Ahuja et al. [1]. This was proposed by Shaw [100] and can be seen to be similar to the 'ruin and recreate' procedure of Schrimpf et al. [98]. The idea is to perturb the solution configuration in an intelligent way by deleting some of the attributes of the solution using some removal strategies and then reintroducing them using some insertion strategies. This has similarities with perturbation methods described earlier except here the removal of many attributes is performed at the same time. Some of the considerations when implementing this method include the number of attributes to remove, the removal strategies adopted and the insertion or repair strategies used. The way these strategies are implemented and developed is critical to the success of the search. As the method is repeated several times some form of learning is worth exploring to efficiently guide the search. For instance [103], [104] introduced interesting new operators and successfully integrated LNS with VNS in an adaptive way. LNS is now becoming a useful tool to solve many complex combinatorial problems.

Random Noise: A related method that perturbs the problem space by introducing random noises to the data is the noising method developed by Charon and Hudry [13]. The idea is to solve the problem with the perturbed objective function values via a local search where at each iteration, or after a fixed number of iterations, the level of perturbation is reduced until it reaches zero

and the problem reverts to the original one. The way the noise is introduced and how randomness is gradually reduced as the search progresses are two key factors that are critical to the success of the method. Relatively recent work by Charon and Hudry [14] explored the self-tuning of the parameters of such a method.

Guided Local Search: This is an adaptive local search which attempts to avoid local optimality and guide the search by using a modified objective function. This objective function contains the original objective and a penalty term which relates to the less attractive features of the configuration of a given local optimum. The local search used can be a simple improvement procedure or a powerful heuristic. In other words, at every locally optimal solution the feature which has a large utility value receives an increase in its unit penalty. For instance, in the case of the TSP, the edges can represent features of the tour and the largest edge of the obtained tour will have its penalty increased. Note that this type of penalty is related to the so-called 'bad' features and not to the amount of infeasibility as usually carried out in constrained optimisation. The way the features are selected and penalised plays an important part in the success of this approach. Voudouris and Tsang [112] provide an interesting review with an emphasis on how to implement this approach when addressing several combinatorial problems.

11.3 Not Necessarily Improving Heuristics

In this section, we discuss those popular heuristics that improve the solution by not necessarily restricting the next move to be an improving move. This enables the search to escape from local optima. However note that allowing such inferior solutions to be chosen needs to be controlled as the search may diverge to even worse solutions. Some of the well-established techniques that are based on this concept are covered here, and include simulated annealing, threshold accepting and tabu search. These methods have been around for many years and are still extremely popular, being used to solve both real-life and academic problems.

11.3.1 Simulated Annealing

The concept of simulated annealing (SA) is derived from statistical mechanics which investigates the behaviour of very large systems of interacting components such as atoms in a fluid in thermal equilibrium, at a finite temperature.

Metropolis et al. [74] studied the simulation of the annealing of solids. This work was built on by Kirkpatrick et al. [59] and later by Cerny [12] who considered how to apply a method based on simulated annealing to large combinatorial optimisation problems. In the context of an optimisation problem, the process of a body cooling is an analogy for a search algorithm seeking a good solution with the aim of approaching the global optimum. The energy function in our case will represent the objective function of the problem. The configuration of the system's particles becomes the solution configuration of the problem. The temperature acts as the control parameter. Cooling too fast resulting in defective crystal is analogous to a neighbourhood search that yields a poor local optimum. Kirkpatrick et al. adopted these analogies for the context of combinatorial optimisation, to explicitly formulate the algorithm that is now widely known as Simulated Annealing (SA). The way in which the temperature decreases is called the cooling schedule and the probability function used is that of Boltzmann's Law (see [74]) which is a negative exponential function. The reasoning behind this choice is that it has the tendency to choose more non-improving solutions initially but as the search progresses this random-based technique will have a smaller probability of selecting inferior solutions meaning the search converges to a high-quality local optimum. For more details see [37], [78], [34] and [33]. A simple implementation of SA is shown in Algorithm 5.

Algorithm 5: Basic Simulated Annealing Algorithm

Step 1. Choose an initial temperature T which is sufficiently high that many worsening moves are likely to be accepted.
Step 2. Create an initial solution s and evaluate its cost value $f(s)$. Set $best = s$
Step 3. Repeat the following n times.
(3a) Choose a neighbouring solution s' and evaluate its cost function.
If $f(s') \leq f(s)$, then set $s = s'$ as it is an improving or equivalent move.
If $f(s') < f(best)$ set $best = s'$
(3b) If $f(s') > f(s)$ then calculate $d = exp(-(f(s') - f(s))/T)$ and if a random number $r(0, 1) < d$, then accept the move and set $s = s'$. Otherwise reject.
Step 4. Reduce the temperature according to some cooling schedule.
Step 5. If the stopping criteria are not met, return to step 3,
Otherwise, stop the search and return $best$.

A disadvantage of simulated annealing is that it is heavily dependent on parameters. These relate to the start and end value of the temperature, the means for reducing the temperature and the number of iterations at each temperature.

There are various methods for reducing the temperature (cooling). The most common is $T_{k+1} = \beta T_k, 0 < \beta < 1$ where the closer β is to 1, the

better the solution quality will normally be but the run time will increase also. Other possible cooling methods include:

$T_{k+1} = T_k - t$ (t = constant) or $T_{k+1} = T_k - t_k$ where t_k is randomly chosen at each iteration k or $T_{k+1} = 1/t(1 + \beta t)$ [69], etc. Typical stopping criteria are a maximum number of iterations is reached, a time limit is attained or the temperature becomes sufficiently small that the likelihood of accepting any worsening moves is likely to be extremely small, so at this stage it is likely that a local optimum has been reached.

The value of the initial temperature (T_0) should be large enough to accept a large number of moves. Therefore an initial phase may be used where random moves are sampled and an appropriate value of the temperature parameter can be calculated to ensure that a given percentage of these moves would be accepted. However, if the value of T_0 is too large, most solutions will be initially accepted which can be a waste of time as the search is in effect performing a random walk. On the other hand, if the value is too small, too many non-improving solutions will be rejected and the method becomes a simple local search method. See [62] for details of some possible implementations, and see [93] for more details. The temperature T_k theoretically needs to be a non-increasing function of iteration k (e.g. $T_{k+1} = g(T_k) \leq T_k$). However it is possible to allow the temperature to remain constant or to increase marginally at a given iteration. One idea is to reset the temperature at higher values after getting stuck in a flat region, say if no improvement is found after a certain number of iterations (i.e. consecutive uphill rejections). The current solution is locally optimal and since the temperature is low, the SA algorithm becomes self-destructive as it restricts the acceptance of less attractive solutions. These non-improving moves are rejected with a probability of almost one. The following resets are commonly used based on the temperature when the best solution was found, say T_{best} and the last temperature reset T_R: $T_{k+1} = T_{\text{best}}$, or $\alpha T_{\text{best}} + (1 - \alpha)T_k$ or $\beta T_{\text{best}} + (1 - \beta)T_R$ (initially $T_R = T_{best}$), with $\alpha, \beta \in [0, 1]$ representing the corresponding weight factors. Some of these resetting schemes are initially given by Connolly [16], [17] and successfully applied by Osman and Christofide [76] to solve the VRP.

One may argue that the update function using such a reset scheme violates the property of $g(T_k)$ as it should be theoretically a non-increasing function of k. This is true if $g(T_k)$ was optimally defined but not heuristically as derived here. Such a statement needs therefore to be relaxed and allowing such flexibility is appropriate. Another way of relaxing such an update would be to incorporate flexibility by allowing the temperature to increase relatively according to the change in the cost function, see [30] and [31]. If a number of

worsening moves are accepted the temperature should be reduced whereas if few worsening moves are being accepted and it appears the search has become trapped in a specific part of the solution space, the temperature should be increased. For more information on SA, see [33].

11.3.2 Threshold-Accepting Heuristics

This heuristic is a simplified version of simulated annealing. It generally involves accepting all moves that fall below some threshold value which varies as the search progresses. This avoids the probabilistic effect of simulated annealing when selecting a non-improving move, as the acceptance decision is entirely deterministic. When implementing threshold acceptance, one again needs to define a starting solution, cost function and neighbourhood. Additionally, the value of the threshold needs to be selected as well as the method for updating it. If the threshold is too high, almost all moves will be accepted and the search will just perform a random walk. If the threshold is too low, no moves will be accepted. Dueck and Scheurer [35] were the first to propose the Threshold Acceptance heuristic formally and they presented empirical studies comparing the results against simulated annealing when solving the TSP. The results were encouraging and Threshold Acceptance has the advantage of requiring fewer parameters and being a simpler method. The method is shown in Algorithm 6.

Algorithm 6: Basic Threshold Accepting Algorithm.

Step 1. Choose an initial threshold value T
Step 2. Create an initial solution s and evaluate its cost value $f(s)$. Set $best = s$
Step 3. Repeat the following n times.
(3a) Choose a neighbouring solution s' and evaluate its cost function.
(3b) If $f(s') \leq f(s)$ or $f(s') \leq T$, then set $s = s'$
(3c) If $f(s') < f(best)$ set $best = s'$
Step 4. Reduce the threshold value according to $T = T * \alpha$ where $\alpha < 1$.
Step 5. If the stopping criteria are not met, return to Step 3,
Otherwise stop the search and return $best$.

An alternative is to record a list of the top non-improving solutions which can then be used to guide the threshold target (the size of such a list can be made constant or dynamically changing). This is similar to SA except that there is flexibility in accepting a non-improving solution deterministically through thresholding instead.

Threshold Acceptance has proved to be computationally efficient when tackling hard combinatorial problems especially routing-based problems (see [107] and [66]). For further references on TA and its implementation, see [52] and [64].

The following two simple but successful variants of TA have proved to be promising and hence are worth mentioning.

Record to record heuristic: This is developed by Dueck [36] where the threshold is based on the relative deviation from the best solution instead of the absolute deviation from the current solution. Therefore moves are accepted if they improve the current move or if the difference between the two costs is below some threshold value. This takes into account the fact that different problems may have very different magnitudes of costs which can be misleading if the classical TA is blindly implemented. Li et al. [66] adopted this approach for solving the heterogeneous vehicle routing problem.

List-based threshold accepting: This approach is developed by Tarantilis et al. [107] where instead of having a threshold value based on the objective function only, a list containing a number of the top solutions is used with its cardinality being the only parameter (M) that needs to be controlled which makes the search easier to implement. During the search, the list is reduced gradually by decreasing the value of M. The authors produced competitive results when testing this scheme on a class of routing problem with a heterogeneous vehicle fleet.

11.3.3 Tabu Search

This approach was proposed by Glover [43] and independently discussed by Hansen [47] as a meta-heuristic optimisation method. Tabu Search (TS) concepts are derived from artificial intelligence where the intelligent use of memory helps to exploit useful historical information. Tabu Search is a best acceptance method that escapes from local optima by accepting non-improving moves. However merely accepting worsening moves will lead to the search cycling between a small number of solutions. To avoid this, a tabu status is allocated to those attributes involved in recent moves. The search is not permitted to return to solutions with these attributes for a certain number of moves. Tabu Search (TS) shares with SA and TA the ability to accept non-improving moves and to escape from local optima.

An attribute of a solution is recorded rather than the entire solution as it is more efficient, but it can lead to solutions being classed as tabu even if they have not been visited before. Therefore an aspiration criterion is normally

used which means that a tabu solution can still be accepted if its cost is better than the best solution found so far.

Tabu search can be considered as an aggressive method as it selects the next best move in a deterministic manner. It will converge quickly to a local optimum before starting to accept worsening moves unlike simulated annealing and threshold methods that may only find locally optimal solutions towards the end of the run. TS also takes into account past information of already found solutions to construct short and/or long-term memories. These memories are useful in guiding the search via diversification to explore other regions of the solution space, and intensification to intensify the search within the same vicinity of the current solution. TS, as other metaheuristics, has also the power of searching over non-feasible regions which, in some situations, can provide an efficient way for crossing the boundaries of feasibility. The general method is shown in Algorithm 7.

Algorithm 7: Basic Tabu Search Algorithm.

Step 1. Create an initial solution s and evaluate its cost value $f(s)$. Set $best = s$
Step 2. Repeat the following n times.
(3a) If there exists a tabu solution s' in $N(s)$ for which $f(s') < f(best)$ then
Set $s = s'$ and update Tabu list.
(3b) Else
Choose the best non-tabu solution s' in $N(s)$
Set $s = s'$ and update Tabu list
(3c) If $f(s) < f(best)$ then $best = s$.

A key question is how to define the right restriction or tabu status. For instance in routing, if the best move was to exchange customer i from route R_k with customer j from route R_s, a very restrictive approach would be to say that both customers cannot be allowed to go back to their original routes, respectively, for a certain number of iterations, a tight restriction would be that either customer i or j is not allowed to go back to its original route but one could return, and a less tight restriction would be that both customers are able to go back to their original routes but cannot be allowed to be inserted between their original predecessors and successors.

The time that a solution remains tabu must also be selected. Small values may increase the risk of cycling whereas large values, on the other hand, may overconstrain the search. Additionally a larger tabu list (size $|Ts|$) can be more computationally demanding. The ideal value may be difficult to decide, and will vary from problem to problem and even instance to instance. It can be defined dynamically so that it updates according to observations made as the

search progresses. It can periodically change between a set of values generated randomly within a fixed range at each iteration. This is known as robust tabu as it provides a more flexible approach. Another scheme is to have a rule that the size of the tabu list switches between the two extreme values. This implementation was found to be successful when adopted by Drezner and Salhi [34] for the one-way network design problem. Finally $|Ts|$ can be dependent on the change in the cost function for that selected move. For instance for the p-median problem, a functional setting was successfully adopted by Salhi [88].

The construction of such functions is seldom attempted in the literature as it is challenging but it is an interesting idea as it incorporates both learning and problem characteristics in an integrated way.

The size of the tabu list can also be updated dynamically by increasing or decreasing its value as the search progresses. For example, $|T_s|$ could be increased according to $(|T_s| = (1 + \beta|T_s|)$ with $\beta = 0.1$ or decreased by setting $\beta = -0.1$ depending on the number of repetitions, their risk of collision, etc. Collision is usually identified through hashing functions or other forms of identification. This is originally proposed by Battiti and Tecchiolli [4] who named this variant as reactive TS (RTS). This approach is adopted by several researchers including Wassan [115] who successfully addressed a class of routing problems, namely, the vehicle routing with backhauls problem.

The level for which the aspiration level overrides the tabu status is crucial in the search as it defines the degree of flexibility of the method. The most common method as mentioned above is to relax the tabu restriction if a solution produces a better result than the currently best-known solution. This is obviously correct as the search would be missing out on the opportunity to identify a new best solution. Another consideration that arises in some circumstances is what will happen if all of the solutions are either tabu or non-improving solutions. Is it appropriate to look for the first non-tabu solution down the list or to choose one from those tabu top solutions? The elements which will constitute the decision need to include the tabu status of the attribute for that solution, the objective function value (or the change in the objective function) and other factors such as frequency of occurrence during a certain number of iterations and so on. One possible attempt is to use the scheme developed by Salhi [88] who introduced a softer aspiration level which is based on the concept of criticality in the tabu status.

Most tabu search implementations include a Diversification step with the aim of guiding the search to new, different parts of the solution space. Diversification moves use some form of perturbation (jumps) to explore new

regions. It uses long-term memory to guide the search out of regions which appears to be either less promising according to the results provided from the short-term memory, or have not been explored previously. The use of diversification is nowadays embedded into most metaheuristics either as a post optimisation step or as a perturbation step as in large neighbourhood search.

Alternatively intensification can be used which stimulates moves to go to a nearby state (neighbourhood) that looks myopically good. This uses short-term memory as it observes the attributes of all performed moves. This is usually achieved by a local search or a series of local searches where the focus of the search is on promising regions of the solution space. In brief, intensification is aimed at detecting good solutions and performing a deeper exploration.

An important question is how to decide when it is better to carry out an intensification and when it is time to perturb the solution and activate diversification. What is the right balance between intensification and diversification? None of these questions are straightforward as these are related to the problem characteristics, the power of the local search used and the type of perturbation the overall search is adopting.

11.4 Population-Based Heuristics

In this section, we present those methods that generate a set of solutions from one iteration to the next instead of only one solution at a time. We give an overview of the most commonly used approaches in this class which include genetic algorithms, ant systems, bees algorithm and particle swarms. For completeness, we also briefly mention other ones such as path relinking, scatter search, harmony search, heuristic cross-entropy, artificial immune systems and the psycho-clonal algorithm.

11.4.1 Genetic Algorithms

Genetic algorithms (GAs) were initiated to mimic some of the processes observed in natural and biological evolution. This approach was initially developed by John Holland and his associates at the University of Michigan in the 1970s. The method was formally introduced for the context of solving general optimisation problems by Holland [50] and Goldberg [45].

In brief, GA is an adaptive heuristic search method based on population genetics. A GA consists of a population of chromosomes (set of solutions) that evolve over a number of generations (iterations) and are subject to genetic operators (transformations) at each generation. The initial population is typically generated at random and a fitness function is defined to measure the quality of each solution. Parent solutions are chosen from the population, typically with a bias towards selecting high-quality solutions. A crossover operator is defined to combine attributes of the parent solutions to create offspring, and then mutations are used to add random variation to the child solutions. The population is updated according to some rules which usually ensures that better solutions have more chance of surviving into the next generation. The process continues for many generations.

A key decision is the chromosome representation of a solution which may involve a binary, permutation, integer or continuous encoding depending on the nature of the problem. For example for the p-median problem where there are n possible sites for p facilities, a binary representation of length n can be used where position $i = 1$ means the site is used and $i = 0$ means it is not. A permutation representation is more suitable for the TSP where the chromosome can consist of a string of integers listing the cities in the order in which they should be visited. An integer representation may be suited to scheduling problems where the value in position i represents the time job i should start. Thought needs to be given to select the right representation as this can either facilitate or hinder the chromosomes' transformation that will be generated via the crossover and mutation operators.

Crossover operators should ensure that children inherit characteristics of the parent solutions. For binary and integer representations possible crossover operators include 1-point, 2-point and uniform which are illustrated in Fig. 11.4. For 1-point crossover, a random cut point is chosen and the child solutions are made up of the elements before the cut point from one parent, and the elements after the cut point from the other parent. For 2-point crossover, 2 random cut points are selected. The child solutions are made up of the elements before the first cut point and after the second cut point from one parent and the elements between the two cut points from the other parent. Uniform crossover selects each element of the child solutions randomly from either parent. For permutation representations, 1-point crossover is not appropriate as it causes infeasible children due to repeated and missing values. Alternatives for permutation representations include partially mapped crossover and order crossover, which are illustrated in Fig. 11.5. For partially mapped crossover, two random cut points are chosen and mapping systems relate the elements between the cut points. In the example shown the

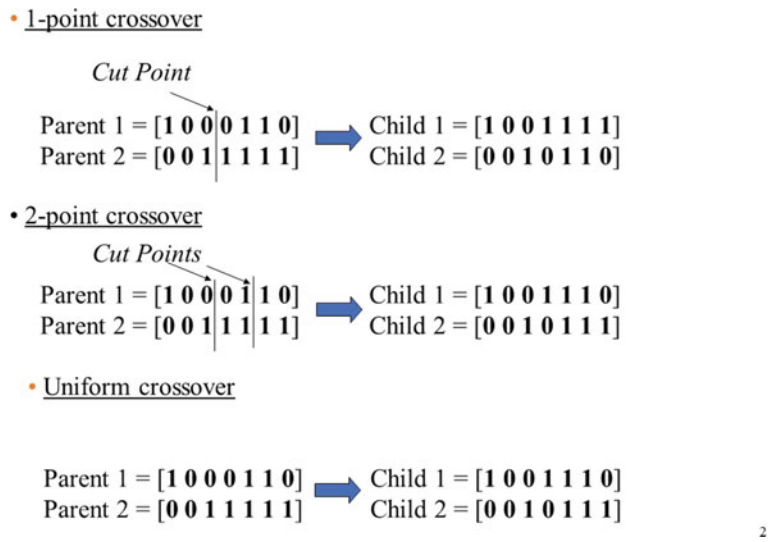

Fig. 11.4 Crossover for binary and integer representations

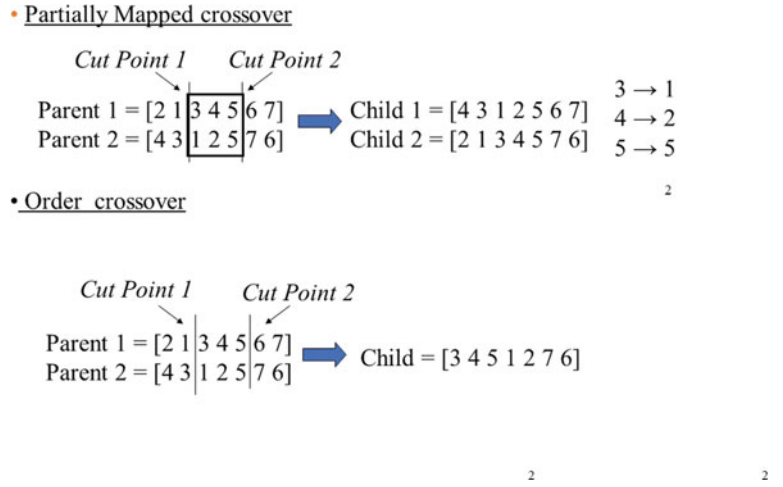

Fig. 11.5 Crossover for permutation representations

mapping systems are [3, 1], [4, 2], [5, 5]. These values are swapped and the remaining values are added if they do not cause any conflict. So for Child 1 for example, 6 and 7 can be added in their positions in Parent 1 but 2 and 1 cannot as they are already present in the child solution. Therefore 2 and 1 are replaced according to the mapping systems by 4 and 3, respectively. For the order crossover, two random cut points are chosen. The child solution is

made up of the values between the two cut points in the first parent, and the remaining values are added in the order they appear in the second parent.

Mutation operators may flip a binary value ($0- > 1$ or $1- > 0$), replace the value in an integer representation or swap values in a permutation representation. A mutation rate λ is typically defined, normally in the range [0.01, 0.1] and then a value is subject to mutation if a random value between 0 and 1 is less than λ.

Other decisions that must be made when using a genetic algorithm include which solutions should be chosen as parents. A commonly used selection scheme is the tournament selection where a subset of the population is chosen randomly and the best of these are selected as parents based on their fitness values (i.e, $f(.)$). There are several other ways to select a chromosome k from population P. Roulette wheel selection is commonly used, where the probability of selecting chromosome k is $\text{Prob}(k) = f(k)/\sum_{r \in P} f(r)$. This simple rule seems to suffer where the population contains some outstanding individuals as these tend to be chosen with a high probability and can lead to premature convergence. However if the population contains solutions that are equally fit, then there is insufficient selection pressure. Other selection rules are based on linear ranking, power and exponential ranking, among others. One interesting approach would be to sort the chromosomes according to their fitness, then construct in addition to the top chromosome, three groups consisting of good, not so good and mediocre or bad chromosomes with respective sizes n_1, n_2 and n_3, respectively. The weights of the three respective groups at generation t say $\alpha_1(t) > \alpha_2(t) > \alpha_3(t) > 0$ with $\sum_{j=1}^{3} \alpha_j(t) = 1$ can be defined. We can also assume that the first weight $\alpha_1(t)$ is a non-decreasing function of t with $\lim_{t \to \infty} \alpha(t) \to 1$ while the other weights converge towards zero. Using the roulette wheel, the group will be selected based on these weights and then a chromosome within that group is chosen randomly for crossover or mutation, etc.

It is also important to provide diversity and hence an opportunity for improvement is possible through some form of migration. In a GA, a small number of completely new chromosomes which are generated either randomly or constructed can be injected into the population to provide diversity from time to time. The number of injected chromosomes and when the injection takes place are issues that deserve careful investigation. Usually the injection starts being active once the GA shows some form of stagnation and then from that point onward injection is activated periodically or adaptively depending on the behaviour of the overall results.

Figure 11.6 illustrates the new combined generation scheme which includes both the flexible reproduction and the effect of immigration which

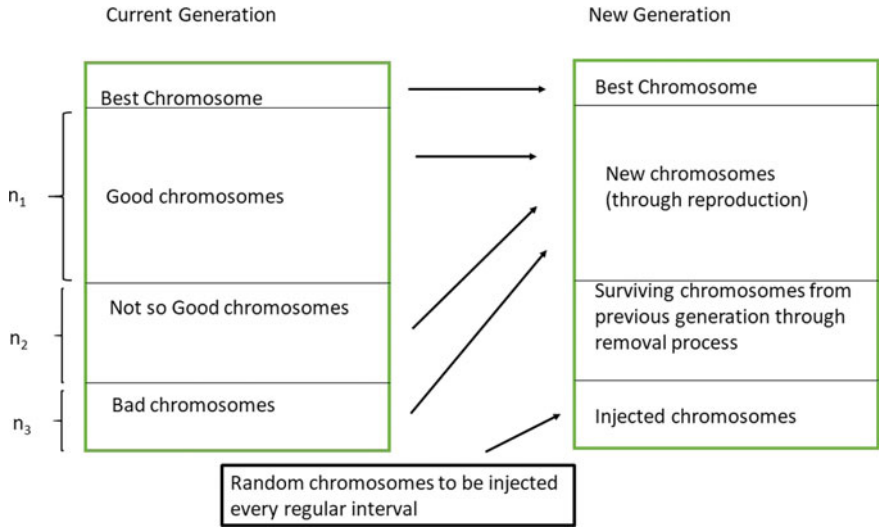

Fig. 11.6 Injection of chromosomes and new generation composition (adapted from [93])

was successfully explored by Salhi and Gama [91], and Salhi and Petch [86] for a class of location and routing problems, respectively. More details can also be found in the recent book on heuristic search in [93].

GAs are powerful at exploring the wider space by identifying promising regions, but lack the fine mechanisms to pinpoint the exact local minima (maxima) that may be required. Therefore many Genetic Algorithms in literature are actually Memetic Algorithms, which are hybrids of GAs and local search. The initial population is produced but is then subject to local search so the population consists of a set of local optima. The algorithm continues as normal but each child that is produced is also subject to descent. In this way, better solutions may be produced but at a cost of additional computational time.

11.4.2 Ant Colony Optimisation

Ant Colony Optimisation (ACO) is a meta-heuristic inspired by the observation of the behaviour of real-life ant colonies, in particular the way in which real ants find the shortest path between food sources and their nest. Ants create paths from the nest to their food and leave pheromone trails which influence the decisions of other ants as to which path to take. The effect of this is the pheromone trail will build up at a faster rate on the shorter paths.

This will influence more ants to follow the shorter paths due to the fact that the ants prefer to follow a path with a higher pheromone concentration. As a greater number of ants choose the shorter path, this in turn causes a greater level of pheromone and hence encourages more ants to follow the shorter path. In time, all ants will have the tendency to choose the shorter path. In other words, the greater the concentration of the pheromone on the ground, the higher the probability that an ant will choose that path. Good quality solutions are developed as a result of this collective behaviour of the ants. For more information see [21].

This real-life behaviour of ants has been adapted to solve combinatorial optimisation problems, as initially proposed by Dorigo et al. [29]. Ant system algorithms employ a set of agents, known as ants, who search in parallel for good solutions using a form of indirect communication. The artificial ants co-operate via the artificial pheromone level deposited on arcs which is calculated as a function of the quality of the solution found. The amount of pheromone an ant deposits is proportional to the quality of the solution generated by that ant helping direct the search towards good solutions. The artificial ants construct solutions iteratively by adding a new node to a partial solution exploiting information gained from past performance using both pheromone levels and a greedy heuristic. A local search could also be introduced to improve the solutions. Once that is completed, a global updating of the pheromone trail levels is activated to reinforce the best solutions by adding an extra amount of pheromone to those arcs of the best solutions. This amount can be based on the length of the best tour as well as the number of ants that produced that best tour.

A variant of ACO named Ant System (AS) was initially proposed by Colorni et al. [15] to solve the travelling salesman problem. The algorithm is illustrated in Algorithm 8. Assume the TSP contains n cities and distance matrix $d(i, j)$ defines the distance between each pair of cities (i, j).

Algorithm 8: Basic Ant System Algorithm.

Step 1. Initialise the placement of M ants and their respective pheromone trail matrix $\tau(i, j)$ to a constant value c where $c > 0$.

Step 2. While none of the stopping criteria witin the inner loop is met, for each ant $(k = 1, \ldots, M)$ perform the following steps:

(2a) Form the complete solution configuration by keep selecting the next customer to visit probabilitistically using the formula: $\tau(i, j)^{\alpha} * \eta(i, j)^{\beta} / \sum_k \tau(i, k)^{\alpha} * \eta(i, k)^{\beta}$ where η_{ij} is known as the visibility and is the greedy cost of going to unvisited city j from current city i to the tour and equals $1/d(i, j)$. α and β are the respective weights on the trail and the visibility.

(2b) Apply a local search if needed (this can be optional, applied once in a while or adaptively using a learning scheme)

(2c) Record the objective function value for each ant $(L_k, k = 1, \ldots, M)$.

(2d) Use the local update of the trail matrix according to the formula $\tau(i, k) = (1 - \rho)\tau(i, k) + \sum_{k=1}^{n} \Delta\tau^k(i, k)$ where the amount of trail added by ant k, $\Delta\tau^k(i, j) = Q/L_k$ for each path (i, j) included in solution k with Q and ρ being the correction and the evaporation factors respectively.

Step 3. If the stopping criteria for the outer loop is met then stop, otherwise

(3a) Apply the global updating of the pheromone trail by adding a constant $\gamma > 0$ to those arcs that belong to the best solution configurations. For example, $\gamma = N_{best} \propto L_{best}$ where N_{best} represents the number of ants that produced the best objective function value L_{best}.

(3b) Return to Step 2.

Colorni et al. also evaluated a trail update rule which add a constant value to the trail which is independent of the total distance of the tour. This made little difference to solution quality. They also showed that it was better to update the trail matrix after the entire tour had been produced rather than after each construction step.

In addition, an elitism strategy that provides a strong reinforcement to the tour corresponding to the global best solution may be used. Arcs belonging to the global best tour have their pheromone increased by a quantity that favour the number of ants that produce the best tour and the length of the global best tour. The consequence of the global updating rule is that arcs belonging to short tours and arcs which have been used frequently are favoured by receiving a greater amount of pheromone. Dorigo and Gambardella [27] proposed updating the pheromone based only on the global best-found solution. Stutzle and Hoos [102] introduced other variants such as the Max-Min ant system, known as MMAS. Their method is similar to AS using the same state transition rule in the selection process with some modifications, the main one is to restrict the pheromone trail values to be in a certain interval $[\tau_{\min}, \tau_{\max}]$. The aim is to reduce the risk of stagnation and make sure that certain choices do not become either dominant with high trail values, or have such small trail values that the probability of them being selected

is approximately zero. It should be noted that although the idea of introducing restrictions on the trail values is interesting as it avoids stagnation and provides a chance for less visited solutions to exist, the choice of these additional parameters $\tau_{\min}$ and $\tau_{\max}$ need to be made appropriately. ACO already has many parameters including the parameters α and β that balance the emphasis on trail and visibility, the number of ants, the number of cycles and the evaporation rate.

Bullnheimer et al. [9] introduced another modification to AS by ranking the ants according to their tour length, and using the ranking to weight the amount of pheromone each ant contributes. Wade and Salhi [113] investigated the use of ACO in a class of VRP by incorporating a visibility factor based on the remaining load in the selection rule, the frequency of occurrence of a given arc in the local updating rule and both elitism and ranking are combined to yield a compromise global updating rule. For more information, references and applications on ant systems in general, the review paper [28] and the book [93] can be useful references.

11.4.3 The Bee Algorithm

This optimisation algorithm is inspired by the behaviour of swarms of honey bees when searching for their food. It shares some similarities with the way the ants behave. The best example of individual insects to resolve complex tasks is by the collection and processing of nectar. Each bee finds sources of the nectar by following another bee which has already discovered a patch of flowers. Once the bee returns to the hive with some food, the bee can either abandon the food source and then becomes again an uncommitted follower, continue to forage at the food source without recruiting new bees, or recruit more bees and return to the food source.

The basic bee algorithm, originally developed for continuous optimisation problems, can be found in [81]. In brief, the bees are first assigned to some chosen sites with the idea that those sites that attract more bees are considered more promising. This is repeated several times till a certain stopping criterion is met. The way a site is considered attractive is the most important point. This is measured by the fitness function of each bee at that site where the top sites will receive more bees either in an equal number or proportionally to their fitness.

Another popular and related bee algorithm is the Artificial Bee Colony (ABC) algorithm proposed by Karaboga [56]. The ABC algorithm is inspired by the intelligent foraging behaviour of a swarm of honeybees. The foraging

bees are classified into three categories: employed, onlookers and scouts. All bees that are currently exploiting a food source are classified as employed and they provide information to the waiting bees (onlooker bees) in the hive about the quality of the food source sites which they are exploiting. Onlooker bees wait in the hive and decide on a food source to further exploit based on the information shared by the employed bees. Scouts search the neighbouring areas to find a new food source. Scout bees can be visualised as performing the job of exploration or diversification, whereas employed and onlooker bees can be considered as performing the job of exploitation or intensification.

The ABC algorithm is an iterative algorithm that starts by assigning all employed bees to randomly generated food sources (solutions). At each iteration, every employed bee determines a food source in the neighbourhood of its currently associated food source and evaluates its nectar amount (fitness). If the new fitness value is better than the previous one then that employed bee moves to this new food source, otherwise it retains its old food source. When all employed bees have finished this process, the bees share the nectar information of the food sources with the onlookers, each of whom selects a food source according to the food quality. This scheme means that good food sources will attract more onlookers than those of poorer quality. After all onlookers have selected their food sources, each of them determines a food source in its neighbourhood and computes its fitness. If an onlooker finds a better fitness value, it changed the employed food source with this new information. However, if a solution represented by a particular food source does not improve for a predetermined number of iterations then that food source is abandoned by its associated employed bee which becomes a scout (i.e. it will search for a new food source randomly). The whole process is repeated until the termination condition is met. The method is shown in Algorithm 9.

Algorithm 9: Basic Bee Colony Algorithm.

Step 1. Perform the following
(1a) Initialise the number of bees B and construct B solutions at random. These are the initial food sources (X_i, $i = 1, \ldots, B$). Evaluate the fitness of each, say $F(X_i)$.
(1b) Initialise T_{max} as the maximum number of iterations and L_{max} as the maximum number of successive iterations without improvement.
(1c) Set the number of successive iterations without improvement = 0 for all bees i.e. $L_i = 0 \forall i = 1, ..B$ and set the neighbours of X_i, say $S_i = N(X_i) = \emptyset$.
Step 2. Repeat the following until the stopping conditions are met
(2a) For each food source, X_i, find $X_i' \in N(X_i)$.
If $F(X_i') < F(X_i)$ set $X_i = X_i'$ and $L_i = 0$. Otherwise let $L_i = L_i + 1$
(2b) For each onlooker, choose a food source X_i randomly; find $X_i' \in N(X_i)$ and set $S_i = S_i \cup X_i'$
(2c) For each food source X_i and $S_i \neq \emptyset$, find the member of S_i with the lowest cost = X_i'.
If $F(X_i') < F(X_i)$ then set $X_i = X_i'$ and $L_i = 0$, else set $L_i = L_i + 1$
(2d) For each food source i, if $L_i = L_{max}$, then select a random move i.e., select $X_i' \in N(X_i)$ and set $X_i = X_i'$. Set $L_i = 0$.

The main steps of the ABC algorithm can be found in Szeto et al. [105] where efficient implementations are presented for solving the vehicle routing problem.

11.4.4 Particle Swarm Optimisation (PSO)

This is a further meta-heuristic method based on the social behaviour of animals, in this case birds or fish. This evolutionary stochastic heuristic is introduced by Kennedy and Eberhat [58] where a population of individuals which searches a region of solutions that is recognised as promising is called a swarm and individual solutions are referred to as particles. It is interesting to stress the similarities which exist in the classical non-linear numerical optimisation techniques where the next point is based on the previous point and the displacement. In other words, the new point at iteration $(k + 1)$ lies along the direction S_k from the previous point X^k with an optimal step size λ_k (i.e. $X^{k+1} = X^k + \lambda_k S_k$). In these numerical optimisation methods the aim is to design a suitable direction and then to derive optimally or numerically the value of the step size. See for instance the classical textbook in numerical optimisation by Fletcher [41] for further information. Here, a particle i (at position p_i) is flown with a velocity V_i through the search space, but retains in memory its best position ($\overrightarrow{p_i}$). In the global PSO each particle, through communication, is aware of the best position of the particles of the swarm ($\overrightarrow{p_g}$). At a given iteration k, the position of the ith particle ($i = 1, ..., n$) is updated as follows $X_{ik} = X_i(k - 1) + V_i^k$. The velocity (or displacement)

is defined as a linear combination of three velocities, namely, (i) the velocity at the previous iteration, (ii) the velocity with respect to the best position of this particle up to this iteration and (iii) the velocity with respect to the global best position of all particles up to this iteration.

PSO has been found to be suitable for solving combinatorial optimisation and especially unconstrained global optimisation problems. As the swarm may become stagnated (early convergence) after a certain number of iterations, a form of diversification or perturbation is often recommended. One way would be to introduce a simple chaotic perturbation that adds diversity to the system and avoids the search from getting stuck.

11.4.5 A Brief Summary of Other Population-Based Approaches

In this subsection, other population-based metaheuristics are briefly described in turn.

Cross-Entropy Based Algorithms (CE): This is an iterative population-based method made up of two steps that are applied in sequence until a stopping criterion is met. These steps are (i) the generation of feasible solutions pseudo-randomly based on a probability distribution representing the frequency of the occurrence of the attributes of a given solution, and (ii) the distribution is then updated. The idea is that this adaptive technique will have the tendency to estimate and learn better probabilities and hence generate better solutions. CE was initially presented by Rubinstein [84] for estimating rare events (financial risk, false alarms, etc) and then Rubinstein and Kroese [85] gave a formal description of this approach and presented its uses in solving combinatorial optimisation problems.

Scatter Search (SS): This is proposed by Glover et al. [44] with the aim of constructing new solutions by combining parts of existing solutions. This method can be considered as one of the earliest steps towards deep learning which is discussed in the next chapter. The idea is to generate a large number of diverse trial solutions which are then improved. The set of these solutions needs to be of high quality while being diverse. A subset of these solutions are classed as the reference solutions; these are typically the best ones but some additional solutions may be included to increase the diversity of the reference set. It could for example consist of 10% of the entire solution set. New solutions are constructed from the solutions in the reference set and these are then improved. Any new solutions that outperform solutions in the reference set are added to it. The process continues until the reference set no longer improves. The main steps of the SS can be found in Marti et al. [72].

Harmony Search (HS): This was originally proposed by Geem et al. [42] to imitate the success of music players in their improvisation when searching for a music harmony that is pleasing to the ear. Such a harmony is made up of a combination of sounds at certain pitches played by different instruments. The aim of the musician is to identify the best pitch for each sound so that when combined they make an excellent harmonious noise. The analogy with optimisation can be seen as follows: A given harmony relates to a given solution configuration, the sounds of the instruments represent the decision variables, their respective pitches are the values of the decision variables, and the quality of the harmony is the objective function value. Each practice by the musician(s) represents the iteration or the generation number. At each practice the musician tries to identify new pitches based on the ones he/she remembers to be of good quality, known as the HS memory, while introducing some extra changes to create a new harmony. This process is repeated until the best harmony already discovered can no longer be improved. HS is a population-based approach where the memory contains the pool of harmonies, similar to the population of chromosomes in GA. However the way the attributes of a new harmony (new solution) are constructed does not depend on two parents only as in GA but on all previously found harmonies. The obtained solution (harmony) is then adjusted for possible improvement with a certain probability. HS relies on its parameter values that are set at the outset. The first attempt to enhance HS was made by Mahdavi et al. [71] followed by a further enhancement by Pan et al. [80].

Artificial Immune Systems (AIS): The immune system's aim is to defend us against diseases and infections. It recognises antigens using immune cells which are known as B-cells whose jobs are to circulate through the blood continuously watching and waiting to encounter antigens (foreign molecules of the pathogens). Each antigen has a particular shape that can be recognised by the receptor of the B cell. AIS is a fast emerging method in some applied areas of computer science such as data analysis and data mining, pattern recognition and has now been adapted to the area of optimisation. In other words, when a pathogen invades the organism it was observed that a number of immune cells which recognise the pathogen will reproduce in large numbers, known as clones. This process is known as reinforcement learning. The clones are then diversified using two methods: (i) a high mutation rate (known as hyper-mutation) which is performed by introducing random changes, and (ii) receptor editing which aims to remove the less attractive antibodies (poor solutions) and replaced them with new ones. In this way, those cells that bind with their antigens are multiplied whereas the others are eliminated following the survival of the fittest. In addition, some of the

successful ones are also kept in memory to face future similar invaders. This concept is similar to the intensification of the search in promising regions whereas the hyper-mutation acts as a diversification strategy. In brief, AIS shares some similarities with GA with the exception that there is no crossover but just a hyper-mutation.

Psycho-Clonal Algorithm: This meta-heuristic was initially developed by Tiwari et al. [110] and it is based on the artificial immune system (AIS) as discussed earlier (mainly based on the clonal selection) and the theory of the hierarchy of social needs as proposed by Maslow [73]. This hierarchy is composed of five levels where the lowest level A refers to the physiological needs (each antibody represents a solution), level B refers to safety needs (the evaluation of the objective function), level C refers to the social needs (the best solutions are selected and cloned proportionally to their objective function value), level D refers to the growth needs (diversification used to generate new solutions via hyper-mutation) and finally the highest level E refers to the self-actualisation needs (the best solutions are chosen to be part of the new population including the injection of new ones). This approach can be considered as a mixture of AIS and guided GA where the management of the population is maintained in a guided way based on the quality of cloning whereas diversity is controlled through the hyper-mutation.

There are numerous examples of heuristics being hybridised with other heuristics. Researchers are only limited by their imaginations as to what is possible. These are discussed in detail in the following chapter. Some hybrids have become extremely common. For example memetic algorithms combine a genetic algorithm with a hill climber meaning all members of the population are local optima according to the neighbourhood definition used. This helps the genetic algorithm identify high-quality solutions but adds considerably to the run time. Similarly a hill climber is often added to Ant Colony Optimisation.

11.5 Some Applications

In this section we present some real-life applications that are addressed by meta-heuristic approaches. There are numerous applications of heuristic and metaheuristic so we can only provide a small overview of a few problems. However we hope this gives an idea of the wide range and variety of problems that are solved in practice.

11.5.1 Radio-Therapy

One of the techniques adopted in treating cancerous tumours is the intensity modulated radiotherapy treatment (short for IMRT). This consists of sending a dose of radiation to the cancerous region with the aim to sterilise the tumour while avoiding damage to the surrounding healthy organs and tissues. This is performed by defining the number of angles, their respective angles and the intensity chosen for the radiation beams at each of these angles. For instance, Bertsimas et al. [5] present a hybridisation of a gradient descent and an adaptive Simulated Annealing method. The initial solution is generated by solving an LP based on using equi-spaced angle beams. Their approach is tested on real-life pancreatic cases (kidneys, liver, stomach, skin and pancreas) at the Massachusetts general hospital of Boston, USA. A case study dealing with patients with head-and-neck tumours at the Portuguese Institute of Oncology in Coimbra is conducted by Dias et al. [24] who adopted a hybrid Genetic Algorithm with a Neural Network. Here, each chromosome is binary and represented by 360 genes, one for each possible angle, with the angles selected represented by 1. Also, Dias et al. [25] proposed Simulated Annealing, with a dynamically adjusted neighbourhood (in terms of angles), and successfully tested it on the same case study in Coimbra. Their results suggest that a reduced number of angles (and hence less technical adjustment) is required and an improvement in organ sparing and coverage of the tumours is observed.

11.5.2 Sport Management

A variety of metaheuristics have been used to schedule fixtures across many different sporting activities. Wright [117] uses Tabu Search to schedule the English county cricket fixtures, Willis and Terrill [116] opt for Simulated Annealing to schedule Australian state cricket, Costa [19] adopts an evolutionary Tabu Search to schedule National Hockey League matches in North America while Thompson [109] also uses Tabu Search to schedule the 1999 Rugby World Cup. Wright [119] produces schedules for New Zealand cricket matches using Simulated Annealing whereas Kendall [57] uses a form of local search to schedule English football fixtures over the Christmas period.

11.5.3 Educational Timetabling

Constructing timetables including examination timetables at schools, colleges and universities can be a hugely time-consuming and difficult task if performed manually. Computer systems that incorporate some form of heuristics are nowadays used frequently and reduce the time required to produce solutions considerably. For instance, Wright [118] constructs a tool that incorporates Tabu Search for a large comprehensive school of over 1400 pupils and 80 teachers in Lancashire, England. Dowsland and Thompson [31] construct an examination timetable for the University of Swansea using Simulated Annealing. The examination timetabling problem is also solved by Di Gaspero and Schaerf [26] using Tabu Search and by Pillay et al. [82] using a Genetic Algorithm. The problem of optimising lecture timetables is considered by, amongst others Borchani et al. [6] using Variable Neighbourhood Descent, Corne et al. [18] using evolutionary algorithms and Basir et al. [3] using Simulated Annealing. For more information and references therein on educational timetabling, see [65].

11.5.4 Nurse Rostering

This problem plays an important part in efficiently managing the personnel at a hospital. The aim is to balance the workforce workload while providing flexibility and satisfying preferences whenever possible leading to a reduction in stress, an increase in staff satisfaction and a happier working environment. Dowsland and Thompson [32] integrate ideas from knapsack problems, network flow models and Tabu Search to construct an efficient computer software tool to solve the nurse rostering problem in a large UK hospital in Wales. Aickelin and Dowsland [2] use an indirect Genetic Algorithm to produce nurse rosters. Burke et al. [11] used Variable Neighbourhood Search to solve the same problem. For more information on this area see the review papers by [10] and [111].

11.5.5 Distribution Management (Routing)

Planning routes by efficiently scheduling the sequence of the customers as they are served and in some cases determining strategically the right vehicle fleet constitutes a huge component of logistic costs (in the range of 30%) and therefore any improvement gained will provide the company with a competitive edge over its competitors. For instance, Semet and Taillard [99] used

Tabu Search to solve a real-life distribution problem in Switzerland leading to a 15% reduction in cost. Rochat and Semet [83] developed a Tabu Search approach for a pet food company having 100 farms and stores leading to about 16% a cost saving. Brandao and Mercier [7] also used Tabu Search for the multi-trip problem at a British biscuit company in the UK. Threshold Acceptance was adopted by Tarantilis and Kiranoudis [107] to schedule the fresh meat distribution with heterogeneous fleet in a densely populated area of Athens. Tarantilis and Kiranoudis [108] used a two-phase approach based on Large Neighbourhood Search for both a dairy and a construction company in Greece. The delivery of blood products to Austrian hospitals for the blood bank of the Austrian Red Cross for Eastern Austria was conducted by Hemmelmayr et al. [51] using a combination of integer programming and Variable Neighbourhood Search.

11.5.6 Location Problems

There is always a challenge in deciding where to locate something which may require a massive investment such as plants and warehouses, consolidation points and in some cases less expensive equipment that are required in large numbers. For instance, when it comes to locating emergency facilities such as police stations, fire stations etc, the aim is to locate the facilities in such a way that the longest time to reach the customer is minimised. This type of problem is known as the p-centre problem where the parameter can be changed for scenario analysis purposes. Pacheto and Casado [79] adopted Scatter Search to locate a number of geriatric and diabetic health care clinics in the rural area of Burgos in Spain. Lu [68] implemented Simulated Annealing to locate urgent relief centres in Taiwan to respond to a major earthquake. Cunha and Silva [20] presented an efficient configuration of such a hub and spoke network for one of the top ten trucking companies in Brazil using a Genetic Algorithm. Also, electricity providers seek to locate their large number of protection devices (costing approximately £10K each) on their tree network to protect the users from having an electricity cut in the case of big storms, etc. James and Salhi [53], [54] explored this unusual network location problem for the UK Midland Electricity Board using a constructive heuristic and Tabu Search.

11.5.7 Chemical Engineering

In several industries such as pharmaceutical, wastewater treatment, biotechnology, the control and the regulation of the pH value, which needs to be around 7, is critical. The modelling of pH control is an important issue which turns out to be an operational decision problem that fits into the class of global optimisation. There is a good amount of research into advanced non-linear control techniques but in practice linear control techniques are usually adopted due to their simplicity and robustness. Research on how to monitor some of the parameters that control the pH is carried out for instance by Mwenbeshi et al. [77] who adopted a powerful GA implementation to intelligently control and model pH in reactors using a lab-scale pH reactor. In this study, a strong base, namely sodium hydroxide is used to neutralise the process made up of four acids whose levels need to be determined in real time as the change in the pH in the reservoir keeps changing all the time. Interesting studies along this area can also be found in their references therein.

11.5.8 Civil Engineering Applications

The design of water distribution networks is very important and costly in the area of civil engineering. This can be seen as a hydraulic infrastructure composed of several pipes of different diameters, hydraulic devices with various powers and different reservoirs. The aim of the problem is to determine the minimal diameter for each pipe in such a way that the total cost is minimised and appropriate water pressure is reached at each of the nodes of the network. HS was tested on this complex non-linear problem based on the water distribution network of Hanoi in Vietnam by Geem et al. [42]. A real coding GA that incorporates neighbourhood reduction with several crossover and mutation operators was proposed by Kadu et al. [55] for the same case study. Reservoir management is also one of the key aspects in water resource planning. Each reservoir has several conflicting objectives as well as different operating rules and operating policies due to the land or the cities around it. The aim is to determine the right policy among a large set of possible ones at a given period. For instance two basic conflicting objectives are the minimisation of the lack of irrigation against the maximisation of the generation of electricity (e.g. hydropower generation). The problem is transformed into a weighted multi-objective approach and solved efficiently using an adaptation of Particle Swarm Optimisation by Kumar and Reddy [61]. The same authors a year earlier in 2006 [60] put forward an approach based on Ant Colony Optimisation to solve the multi-purpose reservoir problem.

There are many applications in other areas of engineering such as electrical, chemical, mechanical, environmental and civil engineering where evolutionary methods including GA are commonly used. See for instance the edited book by Dasgupta [22] which can be a useful and informative addition to the reader.

11.6 Conclusion and Research Issues

In this final section we summarise our findings and provide some research avenues that could be worth pursuing.

11.6.1 Conclusion

In this chapter several heuristic-based techniques that are used in solving difficult combinatorial and global optimisation are described and their pros and cons are highlighted. The methods range from those that only accept improving solutions such as hill-climbing, Variable Neighbourhood Search and GRASP, to those that accept non-improving ones while incorporating some form of guidance to avoid the risk of diverging and cycling like Simulated Annealing, Threshold Acceptance and Tabu Search. Those techniques that use simultaneously more than one solution at a time, also known as population or evolutionary methods, are also discussed include Genetic Algorithms, Ant Colony Optimisation and Bee Colony Optimisation. As strengths and weaknesses can be found in any heuristic, many modern implementations hybridise a number of these methods, leading to better results overall. Of particular interest is metaheuristics, where exact and heuristic methods are combined.

The efficiency of heuristics depends on several aspects and one key element is the quality of the implementation. Each meta-heuristic can be applied in different ways and reviewing the academic literature to understand which sorts of methods have worked well on particular problems is crucial. Also key is parameter optimisation and here methods that require fewer parameters e.g. tabu search, threshold acceptance may be considered superior to methods that require many parameters such as simulated annealing and ant colony optimisation. Methods that automatically set parameters or dynamically adjust them as the search proceeds may remove this advantage. The coding is crucial also—much time can be saved by using efficient data structures, reducing neighbourhoods, etc.

Heuristic search has made huge advances in the last 25 years and this is likely to continue as problem complexity and size also increase. Advances in computer technology and commercial optimisation software are enabling larger problems to be solved exactly than previous, however there is still a huge need for high-quality heuristic solution methods.

Heuristics have arisen from a variety of applications, from different people's expertise and sometimes just as a by-product of curiosity of some researchers whose original aim was to disprove their usefulness. We believe this less structured area, known by some as a grey research area, will remain for many years to come and will become even greyer and open to more challenges. Heuristics remain the most appropriate and attractive optimisation approaches for tackling many complex combinatorial and global optimisation problems.

11.6.2 Potential Research Issues

An exciting future topic is hybridisation, be it between purely heuristic methods or heuristic and exact methods. This suggestion will be discussed in another chapter in this book [97].

The understanding of how a given method works and then the design of a data structure that incorporates interesting information found during the search so as to avoid re-computing unnecessary calculations is vital. This process though may increase some level of memory and may require an initial fixed cost in terms of development and computation time, but it does often lead to a massive time saving without affecting the solution quality at all. This aspect can also be enhanced further by the construction of effective neighbourhood reduction schemes that helps the search to avoid checking combinations and operations that are unlikely to result in improving the solution. A note of caution here is that the latter schemes will have a considerable saving in computational time and are usually simple to construct, but could affect the solution quality if they happen to be too restrictive. The compromise in the design between a neighbourhood reduction method which is powerful enough (removing as many as possible irrelevant checks) while not excluding promising moves is exciting and hence worth exploring.

Evolutionary heuristics are relatively easy to be parallelised and hence can be used for larger instances and in a practical setting if the computing facilities are available.

The design of adaptive search that dynamically learns and makes use of the obtained information is crucial. This learning mechanism which ought to be continuously or at least periodically updated is then used to pseudo-randomly select at regular intervals the decision rules to be used. These can include

a subset of neighbourhoods to choose from, a number of local searches to be used for intensification purposes, or even the powerful heuristics or exact methods to select from. This kind of search is self-adaptive and also efficient as it uses only what it needs with the expectation that a good solution may be found. This research issue though challenging and practically useful will probably be one of the most popular research areas in the near future as it has the additional benefits of being applicable in several areas ranging from engineering to medicine.

In a related but different aspect, it is well known that most exciting powerful heuristics seem to suffer from parameter tuning. It is therefore worthwhile concentrating on schemes that incorporate ways of reducing the number of parameters or adjusting the parameters' values dynamically and adaptively. This is very welcome and in our view is also one of the ways forward. Results from such studies will provide us with tools that avoid requiring excessive time for fine tuning of the parameters, besides making the heuristic less sensitive to parameter values and hence reliable to use.

References

1. Ahuja RK, Ergun O, Orlin JB and Punnen AP (2002). A survey of very large scale neighbourhood search techniques. Discrete Appl Math 123: 75–102.
2. Aickelin U and Dowsland K (2000). Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. J Sched 3: 139–153.
3. Basir N, Ismail W and Norwawi, N (2013). A simulated annealing for Tahmidi course timetabling. Procedia Technology 11: 437–445.
4. Battiti R and Tecchiolli G (1994). The reactive tabu search. ORSA J Comput 6: 126–140.
5. Bertsimas D, Cacchiani V, Craft D and Nohadani O (2013). A hybrid approach to beam angle optimization in intensity-modulated radiation therapy. Comput Oper Res 40: 2187–2197.
6. Borchani E, Elloumi A and Masmoudi M (2017). Variable neighbourhood descent search algorithms for course timetabling problem: Application to a Tunisian University. Electronic Notes in Discrete Math. 58: 119–126.
7. Brandao J and Mercer A (1997). A tabu search heuristic for the multiple-trip vehicle routing and scheduling problem. Eur J Oper Res 100: 180–191.
8. Braysy O, Hasle G and Dullaert W (2004). A multi-start local search algorithm for the vehicle routing problem with time windows. Eur J Oper Res 159 (3): 586–605.
9. Bullnheimer B, Harlt R and Strauss C (1998). Applying ant systems to the vehicle routing problem. In Voss S, Martello S, Osman IH and Roucairal

C (eds), Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization. Kluwer Academic Publishers, Boston.
10. Burke EK, De Causmaecker P, Berghe GV and Van Landeghem H (2004). The state of the art of nurse rostering. J Sched 7: 441–499.
11. Burke EK, Curtois T, Post G, Qu R and Veltman B (2008). A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. Eur J Oper Res 188 (2): 330–341.
12. Cerny V (1982). A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. J Optim Theory Appl 45: 41–51.
13. Charon I and Hudry O (1993). The noising method—a new method for combinatorial optimization. Oper Res Let 14: 133–137.
14. Charon I and Hudry O (2009). Self-tuning of the noising method. Optimization 58: 1–21.
15. Colorni A, Dorigo M and Maniezzo V (1991). Distributed optimization by ant colonies. In Varela F and Bourgine P (eds) Proceedings of the European Conference on Artificial Life. Elsevier Publishing, Amsterdam, 457–474.
16. Conolly DT (1990). An improved simulated annealing technique for the QAP. Eur J Oper Res 46: 93–100.
17. Conolly D (1992). General purpose simulated annealing. J Opl Res Soc 43: 495–505.
18. Corne D, Ross P and Fang H-L (2005). Fast practical evolutionary timetabling. In: Fogarty T C (ed) Evolutionary Computing. Lecture Notes in Computer Science 865, Srpinger, Berlin, Heidelberg.
19. Costa D (1995). An evolutionary tabu search algorithm and the NHL scheduling problem. INFOR 33: 161–178.
20. Cunha CB and Silva ME (2007). A genetic algorithm for the problem of configuring a hub-and-spoke network for a LTL trucking company in Brazil. Eur J Oper Res 179: 747–758.
21. Daneubourg JL, Aron A, Goss S and Pasteels JM (1990). The self organising exploratory pattern of the argentine ant. J Insec Behav 3: 159–168.
22. Dasgupta D and Michalewicz Z (Eds) (2013). Evolutionary Algorithms in Engineering Applications. Springer, New York.
23. Dasgupta D (Ed) (1999). Artificial Immune System and Their Applications. Springer- Verlag.
24. Dias J, Rocha H, Ferreira B, de Carmo Lopes C (2014). A genetic algorithm with neural network fitness function evaluation for IMRT beam angle optimization. Cent Eur J Oper Res 22: 431–455.
25. Dias J, Rocha H, Ferreira B, de Carmo Lopes C (2014). Simulated annealing applied to IMRT beam angle optimization: a computational study. Physica Medica 31: 747–756.
26. Di Gaspero L and Schaerf A (2001). Tabu search techniques for examination timetabling. In: EK Burke and W Erben (eds) Selected Papers from the Third International Conference on the Practice and Theory of Automated Timetabling. Lecture Notes in Computer Science 2079, 104–117.

27. Dorigo M and Gambardella LM (1997). Ant colony system: a cooperative learning approach to the travelling salesman problem. IEEE Trans Evol Comput 1: 53–66.
28. Dorigi M and Stutzle T (2010). Ant colony optimization: overview and recent advances. In Gendreau M and Potvin JY (eds) Handbook of Metaheuristics (2nd edition). Springer, London, pp 227–264.
29. Dorigo M, Caro G and Gambardella L (1999). Ant algorithms for discrete optimization. Art Life 5: 137–172.
30. Dowsland KA (1993). Some experiments with simulated annealing techniques for packing problems. Eur J Oper Res 68: 389–399.
31. Dowsland KA and Thompson JM (1998). A robust simulated annealing based examination timetabling system. Comp Oper Res 25: 637–648.
32. Dowsland KA and Thompson JM (2000). Solving a nurse scheduling problem with knapsacks, network and tabu search. J Oper Res Soc 51: 825–833.
33. Dowsland KA and Thompson JM (2012). Simulated annealing. In Rozenberg G, Back T and Kok JN (eds) Handbook of Natural Computing. Springer-Verlag, Berlin, pp 1624–1655.
34. Drezner Z and Salhi S (2002). Using hybrid metaheuristics for the one-way and two-way network design problem. Nav Res Logistics 49: 449–463.
35. Dueck G and Scheuer T (1990). Threshold accepting: a general purpose optimization algorithm superior to simulated annealing. J Comput Phy 90: 161–175.
36. Dueck G (1993). New optimization heuristics: the great deluge algorithm and the record-to-record travel. J Comp Phys 104: 86–92.
37. Eglese R (1990). Simulated annealing: a tool for operational research. Eur J Oper Res 46: 271–281.
38. Elshaikh A, Salhi S, Brimberg J, Mladenović N, Callaghan B and Nagy G (2016). An adaptive perturbation-based heuristic: an application to the continuous p-centre problem. Comput Oper Res 75: 1–11.
39. Feo TA and Resende MGC (1989). A probablistic heuristic for a computationally difficult set covering problem. Opns Res Lett 8: 67–71.
40. Feo TA and Resende MGC (1995). Greedy randomized adaptive search procedures. J Glob Opt 6: 109–133.
41. Fletcher R (1989). Practical Methods of Optimisation. John Wiley and Sons, New York.
42. Geem ZW, Kim JH and Loganathan GV (2001). A new heuristic optimization algorithm: harmony search. Simulation 76 (2): 60–68.
43. Glover F (1986). Future paths for integer programming and links to artificial intelligence. Comput Opns Res 13: 533–549.
44. Glover F, Laguna M and Marti R (2003) Scatter search and path relinking: advances and applications. In Glover F and Kochenberger GA (eds) Handbook of Metaheuristics. Kluwer Academic Publisher, London, pp 1–35.
45. Goldberg DE (1989). Genetic Algorithm in Search, Optimization and Machine Learning. Addison-Wesley, New York.

46. Goodman M, Dowsland KA and Thompson JM (2009) A grasp-knapsack hybrid for a nurse-scheduling problem. J Heuristics 15: 351–379.
47. Hansen P (1986). The steepest ascent, mildest descent heuristic for combinatorial programming. Paper presented at the congress on Numerical Methods in Combinatorial Optimization, Capri, Italy.
48. Hansen P, Mladenović N, Brimberg J and Moreno Perez JA (2019). Variable neighbourhood search. In Gendreau M and Potvin JY (eds) Handbook of metaheuristics (latest edition). Springer, Cham, pp 57–97.
49. Hansen P, Mladenović N, Todosijević and Hanafi S (2017). Variable neighborhood search: basics and variants. EURO J Comput Optim 5: 423–454.
50. Holland JH (1975). Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Harbor.
51. Hemmelmayr V, Doerner KF, Hartl RF and Savelsbergh MWP (2009). Delivery strategies for blood products supplies. OR Spec 31: 707–725.
52. Hu TC, Kahng AB and Tsao CWA (1995). Old bachelor acceptance: a new class of non-monotone threshold accepting methods. ORSA J Comput 7: 417–425.
53. James C and Salhi S (2000). The location of protection devices on electrical tree networks: a heuristic approach. J Oper Res Soc 51: 959–970.
54. James C and Salhi S (2000). A tabu search heuristic for the location of multi type protection devices on electrical tree networks. J Com Opt 6: 81–98.
55. Kadu MS, Gupta R and Bhave P (2008). Optimal design of water networks using a modified genetic algorithm with reduction in search space. J Water Res Plan Manage 134: 147–160.
56. Karaboga D and Basturk B (2007). Artificial bee colony optimization algorithm for solving constrained optimization problems. In Melin P, Castillo O, Aguilar L, Kacprzyk J and Pedrycz, W (eds) Foundations of Fuzzy Logic and Soft Computing. Lecture Notes in Computer Science 4529, Berlin, Heidelberg.
57. Kendall G (2008). Scheduling English football fixtures over holiday periods. J Oper Res Soc 59: 743–755.
58. Kennedy J and Eberhault RC (1995). Particle Swarm Optimization. IEEE Int Conf Neural Networks, Perth, Australia, pp 1942–1948.
59. Kirkpatrick S, Gelat CD and Vecchi MP (1983). Optimization by simulated annealing. Science 220: 671–680.
60. Kumar DN and Reddy MJ (2006). Ant colony optimization for multi-purpose reservoir operation. Water Res Manage 20: 879–898.
61. Kumar DN and Reddy MJ (2007). Multi-purpose reservoir operation using particle swarm optimization. J Water Resour Plann Manag 133 (3): 192–201.
62. Laarhoven PJM and Aarts EHL (1987). Simulated Annealing: Theory and Applications. Reidel, Rotterdam.
63. Laporte G, Gendreau M, Potvin J-Y and Semet F (2000). Classical and modern heuristics for the vehicle routing problem. International Transaction in Operational Research 7: 285–300.

64. Lee DS, Vassiliadis VS and Park JM (2004). A novel threshold accepting meta-heuristic for the job-shop scheduling problem. Comp Oper Res 31 (13): 2199–2213.
65. Lewis R (2008). A survey of meta-heuristic based techniques for university timetabling problems. OR Spektrum 30: 167–190.
66. Li F, Golden B and Wasil E (2007). A record-to-record travel algorithm for solving the heterogeneous fleet vehicle routing problem. Comput Opns Res 34: 2734–2742.
67. Lourenco HR, Martin OC and Stutzle T (2010). Iterated local search: framework and applications. In Gendreau M and Potvin JY (eds) Handbook of Metaheuristics. Springer, London, pp 363–397.
68. Lu C (2013). Robust weighted vertex p-center model considering uncertain data: an application to emergency management. Eur J Oper Res 230: 113–121.
69. Lundy M and Mees A (1986). Convergence of an annealing algorithm. Math Prog 34: 111–124.
70. Luis M, Salhi S and Nagy G (2011). A guided reactive GRASP for the capacitated multi-source Weber problem. Comp Oper Res 38 (7): 1014–1024.
71. Mahdavi M, Fesanghary M and Damangir E (2007). An improved harmony search algorithm for solving optimization problems. Applied Mathematics and Computation 188: 1567–1579.
72. Marti R, Laguna M and Glover F (2006). Principles of scatter search. Eur J Oper Res 169: 359–372.
73. Maslow AH (1954). Motivation and Personality. Harper & Sons, New York.
74. Metropolis N, Rosenbluth A, Rosenbluth M, Teller A and Teller E (1953). Equations of state calculations by fast computing machines. J Chem Phy 21: 1087–1092.
75. Mladenović N and Hansen P (1997). Variable neighbourhood search. Comput Oper Res 24: 1097–1100.
76. Osman IH and Christofides N (1994). Capacitated clustering problems by hybrid simulated annealing and tabu search. Int Trans Oper Res 1: 317–336.
77. Mwenbeshi MM, Kent CA and Salhi S (2004). A genetic algorithm based approach to intelligent modelling and control of pH in reactors. Comp Chem Eng 28 (9): 1743–1757.
78. Osman IH and Laporte G (1996). Metaheuristics: a bibliography. Ann Oper Res 63: 513–623.
79. Pacheto JA and Casado S (2004). Solving two location models with few facilities by using a hybrid heuristic: a real health resources case. Comput Oper Res 32: 3075–3091.
80. Pan QK, Suganthan PN, Tasgetiren MF and Liang JJ (2010). A self-adaptive global best harmony search algorithm for continuous optimization problems. Appl Math Comput 216: 830–848.
81. Pham DT, Ghanbarzadeh A, Koc E, Otri S, Rahim S and Zaidi M (2006). The bees algorithm, a novel tool for complex optimisation problems. In Proc

2nd Virtual International Conference on Intelligent Production Machines and Systems, Elsevier, Oxford, pp 454–459.
82. Pillay N and Banzhaf W (2010). An informed genetic algorithm for the examination problem. Appl Soft Comput 10 (2): 457–467.
83. Rochat Y and Semet F (1994). A tabu search approach for delivering pet food and flour in Switzerland. J Oper Res Soc 45: 1233–1246.
84. Rubinstein RY (1997). Optimization of computer simulation models with rare events. Eur J Oper Res 99: 89–112.
85. Rubinstein RY and Kroese DP (2004). The cross-entropy method: a unified approach to combinatorial optimization, Monte Carlo simulation and machine learning. Springer-Verlag, New York.
86. Salhi S (1997). A perturbation heuristic for a class of location problem. J Oper Res Soc 48: 1233–1240.
87. Salhi S (1998). Heuristic search methods. In Marcoulides GA (ed) Modern methods for business research. Lawrence Erlbaum Associates, New Jersey, pp 147–175.
88. Salhi S (2002). Defining tabu list size and aspiration criterion within tabu search methods. Comput Opns Res 29: 67–86.
89. Salhi S and Rand GK (1987). Improvements to vehicle routing heuristics. J Oper Res Soc 38: 293–295.
90. Salhi S and Sari M (1997). A Multi-level composite heuristic for the multi-depot vehicle fleet mix problem. Eur J Oper Res 103: 78–95.
91. Salhi S and Gamal MDH (2003). A genetic algorithm based approach for the uncapacitated continuous location–allocation problem. Ann Oper Res 123: 203–222.
92. Salhi S (2006). Heuristic search in action: the science of tomorrow. In Salhi S (Ed) OR48 Keynote papers. ORS Bath, pp 39–58.
93. Salhi S (2017). Heuristic Search: The Emerging Science of Problem Solving. Palgrave MacMillan.
94. Salhi S, Gutierrez, Wassan N, Wu S and Kaya R (2020). An effective real time GRASP-based metaheuristic: application to order consolidation and dynamic selection of transshipment points for time-critical freight logistics. Expert Syst Appl 158: 113574.
95. Salhi S and Petch R (2007). A GA based heuristic for the vehicle routing problem with multiple trips. J Math Model Algor 6: 591–613.
96. Salhi S, Imran A and Wassan NA (2014). The multi-depot vehicle routing problem with heterogeneous vehicle fleet: formulation and a variable neighbourhood search implementation. Comput Oprs Res 52: 315–325.
97. Salhi S and Thompson J (2021). The new era of hybridisation and learning in heuristic search design. In Salhi S and Boylan J (eds) The Palgrave Handbook of Operations Research. Palgrave, London.
98. Schrimpf G, Schneider J, Stamm-Wilbrabdt H and Dueck H (2000). Record breaking optimization results-using the ruin and recreate principle. J Comput Phys 159: 139–171.

99. Semet F and Taillard E (1993). Solving real-life vehicle routing problems efficiently using tabu search. Ann Oper Res 41: 469–488.
100. Shaw P (1998). Using constrain programming and local search methods to solve vehicle routing problem. In Proceeding CP-98 (Fourth International Conference on Principles and Practice of Constraint Programming).
101. Sorenson K, Sevaux M and Glover F (2018). A history of metaheuristics. In Marti R, Pardalos P and Resende M (eds) Handbook of Heuristics. Springer, 791–808.
102. Stützle T and Hoos HH (2000). MAX–MIN ant system. Futur Gener Comput Syst 16: 889–914.
103. Sze J, Salhi S and Wassan N (2016). A hybridisation of adaptive variable neighbourhood search and large neighbourhood search: application to the vehicle routing problem. Expert Syst Appl 65: 383–397.
104. Sze J, Salhi S and Wassan N (2017). The cumulative capacitated vehicle routing problem with min-sum and min-max objectives: an effective hybridisation of adaptive variable neighbourhood search and large neighbourhood search. Transp Res Part B 101: 162–184.
105. Szeto WY, Wu Y and Ho SC (2011). An artificial bee colony algorithm for the capacitated vehicle routing problem. Eur J Oper Res 215: 126–135.
106. Tarantilis CD and Kiranoudis CT (2002). BoneRoute: an adaptive memory-based method for effective fleet management. Ann Opns Res 115: 227–241.
107. Tarantilis CD, Kiranoudis C and Vassiliadis V (2003). A list based threshold accepting metahauristic for the heterogeneous fixed vehicle routing problem. J Oper Res Soc 54: 65–71.
108. Tarantilis CD and Kiranoudis CT (2007). A flexible adaptive memory-based algorithm for real-life transportation operations: two case studies from Diary and construction sector. Eur J Oper Res 179: 806–822.
109. Thompson J (1999). Kicking timetabling problems into touch. OR Insight 12: 7–15.
110. Tiwari MK, Prakash A, Kumar A, Mileham AR (2005). Determination of an optimal sequence using the psychoclonal algorithm. J Eng Manuf 219: 137–149.
111. Valouxis G, Gogos C, Goulas G and Alefragis P (2012). A systematic two phase approach for the nurse rostering problem. Eur J Oper Res 219: 425–433.
112. Voudouris C and Tsang EPK (2010). Guided local search. In Gendreau M, and Potvin JY (eds) Handbook of Metaheuristcs. Springer, London, pp 321–361.
113. Wade AC and Salhi S (2003). An ant system algorithm for the mixed vehicle routing problem with backhauls. In Resende MG and de Sousa JP (eds) Metaheuristics: Computer Decision-Making. Kluwer, NY, pp 699–719.
114. Wang H, Yao Y and Salhi S (2021). Tension in big data using machine learning: analysis and applications. Tech For Soc Change 158: 120175.

115. Wassan NA (2006). A reactive tabu search for vehicle routing. J Oper Res Soc 57: 111–116.
116. Willis R and Terrill B (1994). Scheduling the Australian state cricket season using simulated annealing. J Oper Res Soc 45: 276–280.
117. Wright M (1994). Timetabling county cricket fixtures using a form of tabu search. J Oper Res Soc 45: 758–770.
118. Wright M (1996). School timetabling using heuristic search. J Oper Res Soc 47: 347–357.
119. Wright M (2005). Scheduling fixtures for New Zealand cricket. IMA J Manag Math 16: 99–112.
120. Zainuddin ZM and Salhi S (2007). A perturbation-based heuristic for the capacitated multisource Weber problem. Eur J Oper Res 179: 1194–1207.