Heuristics: An Overview

Jonathan Thompson

Contents

Introduction . . . 2
Background of Heuristics . . . 5
Greedy Heuristics . . . 7
Improvement Heuristics . . . 8
Descent Methods . . . 10
Multi-Start Methods . . . 11
Simulated Annealing . . . 11
Tabu Search . . . 13
Threshold Acceptance . . . 15
Greedy Randomized Adaptive Search Procedure (GRASP) . . . 15
Variable Neighborhood Search (VNS) . . . 16
Population-Based Methods . . . 17
Genetic Algorithms . . . 17
Ant Colony Optimization (ACO) . . . 20
Bee Colony Optimization . . . 22
Harmony Search . . . 23
Other Heuristic Methods . . . 24
Matheuristics . . . 24
Hyper-Heuristics . . . 25
Conclusions . . . 26
References . . . 27

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

A. J. Kulkarni, A. H. Gandomi (eds.), *Handbook of Formal Optimization*,
https://doi.org/10.1007/978-981-19-8851-6_32-1

Abstract

There are many problems that still cannot be solved exactly in a reasonable time despite rapid increases in computing power, including the Travelling Salesman Problem and the Examination Timetabling Problem. Therefore, there is still a strong need for heuristics and this chapter demonstrates that the search for better heuristics remains a dynamic research topic. An overview is provided of many traditional heuristic methods including Simulated Annealing, Tabu Search, and Genetic Algorithms as well as more recently proposed methods including Harmony Search, Hyper-heuristics, and Matheuristics. Examples are provided for the Travelling Salesman Problem and the Examination Timetabling Problem showing that high-quality solutions can be produced in a reasonable time if a suitable heuristic method is applied appropriately.

Keywords

Heuristic · Meta-heuristic · Optimization · Travelling salesman problem · Examination timetabling problem

Introduction

We live in a time of instant gratification. People are less willing to wait than previous generations who were more likely to save to make a purchase or to queue to buy a product. In modern culture, people borrow in order to make their purchase immediately, and they expect quick delivery of shopping. People also expect instant and accurate information and precise answers to questions. The age of the internet has made all these things possible.

Modern computing has made many amazing achievements possible. A relatively cheap laptop has more computing power than was used to take man to the moon. Therefore, it seems reasonable to expect optimal solutions to the everyday problems faced by society. However even the best current computers cannot solve all problems. This is because of the NP problem – that is for many combinatorial problems, the time required to find the optimal solution increases exponentially as problem size increases. As society advances, problems are becoming both larger and more complex and the rapid increase in computing power that has occurred over the last few years has not solved this problem. Many researchers believe that there is no solution method that will address the problem of rapidly increasing run times as problem size increases.

Combinatorial optimization is concerned with discrete optimization problems. There are numerous examples of such problems including:

- Scheduling problems including scheduling jobs in factories and scheduling sports fixtures.
- Timetabling problems including examination and lecture timetabling and train and bus timetabling.

- Routing problems including routing vehicles as they deliver goods to consumers, and routing robotic arms as they install components on circuit boards.
- Cutting and packing problems including cutting material into pieces of specified sizes while minimizing wastage and packing parcels into bins.

Many variants of these problems lie in the set of NP-complete problems. Small instances of such problems can be solved exactly but for larger problems, a different solution method is required if a solution is to be found in a reasonable amount of time. This does require compromise in terms of solution quality and the user needs to consider how important solution quality is to them. In general, the longer the period available to find a solution, the better quality the solution will be. For example, if a university wishes to produce its summer examination timetable, the problem only has to be solved once per year, so it is reasonable to assign a good amount of time to optimizing the quality of the timetable. The better the timetable, the happier the staff and students will be so the extra solution time is justified. However, if a company makes daily deliveries to customers and therefore needs to deliver quickly and promptly, there is little time to produce an optimal route and incurring 5% additional diesel cost may not be particularly important to the company compared with producing a route quickly.

Therefore, heuristics are frequently used to solve combinatorial optimization problems. Heuristic comes from the Greek word heuriskein that means discover and explore and commonly refers to an approximate method that produces a good but not necessarily optimal solution in a reasonable amount of time. Their use is commonplace and research into promising heuristic methods has been widespread over recent years. Figure 1 shows the number of papers that have included

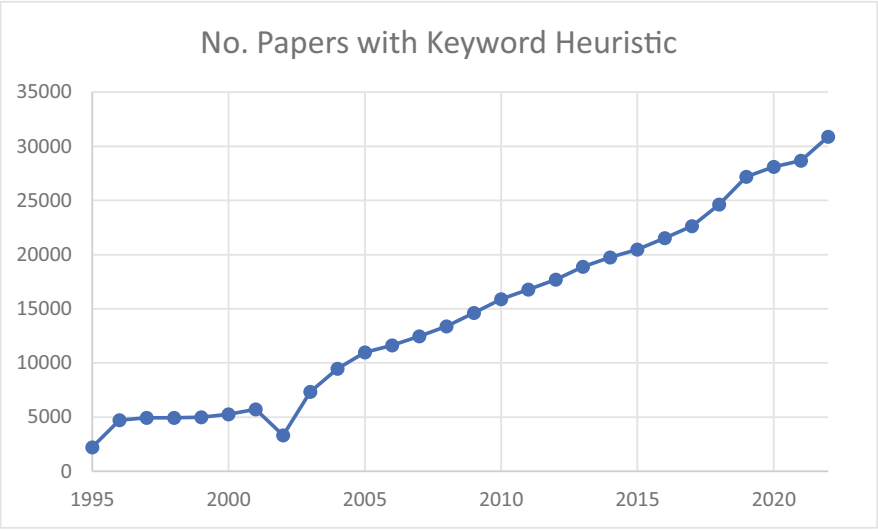

Fig. 1 Number of papers with keyword of Heuristic in Scopus

"heuristic" as a keyword in Scopus in recent years. This shows that heuristics are a dynamic and growing research topic and show no sign of slowing down in popularity.

This chapter will provide an overview of heuristics and describe examples of their application. Heuristics will be divided into three categories:

- Greedy heuristics which construct a solution from scratch.
- Improvement heuristics that take some starting solution and attempt to improve it.
- Population heuristics which construct a set of solutions and attempt to improve them.

We will then focus on two other heuristic methods that have the potential to improve the current best heuristic methods, Hyper-heuristics, and Matheuristics.

A number of general heuristic methods named meta-heuristics have been proposed in recent years. Figure 2 shows how they can be divided into the above categories. This demonstrates that not all meta-heuristic methods fit neatly into one category, for example Genetic Algorithms are a population heuristic, but they have been combined with descent to form Memetic Algorithms. These lie at the intersection of improvement and population heuristics. Ant colony optimization is a construction heuristic that uses a population of ants. Sometimes, a descent phase is included also meaning that ant colony optimization may be considered to lie at the intersection of all three methods.

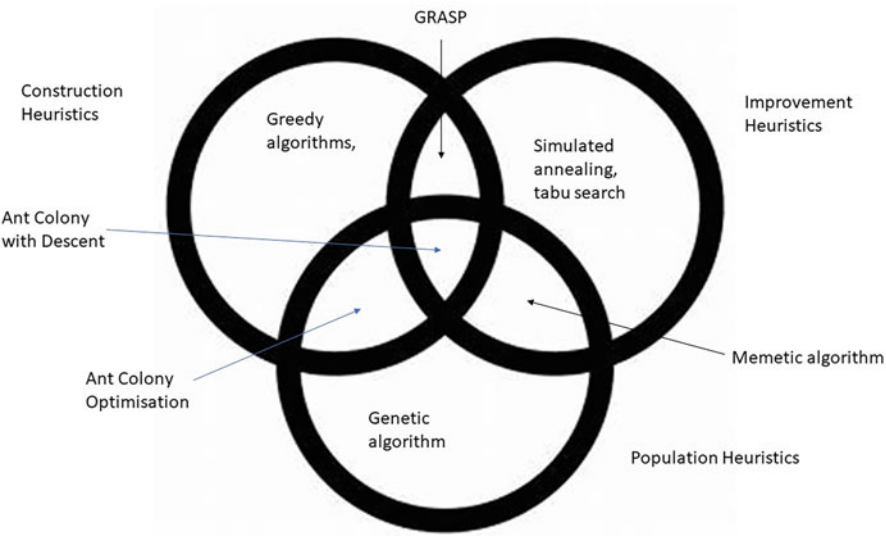

Fig. 2 Classification of meta-heuristic methods

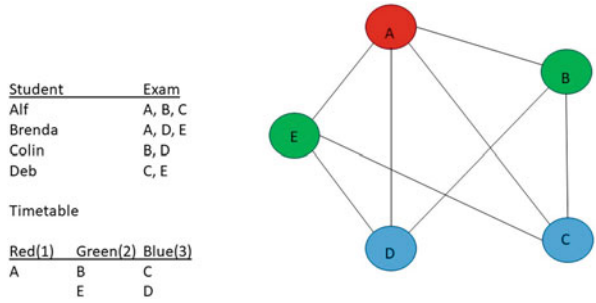

Fig. 3 A graph coloring representation of an exam timetabling problem

When describing heuristics, we will use two example problems:

- The Travelling Salesman Problem (TSP). Arguably the most famous combinatorial optimization problem, this problem involves finding the route that visits a set of cities and returns to the starting point that minimizes the distance the salesman must travel.
- The Examination Timetabling Problem (ETP). This problem involves allocating a set of exams to a set of time periods such that all exams are scheduled, no student has two exams at the same time and various other constraints are satisfied including room capacities. The objective is normally to spread the exams out for students so that they have gaps between their examinations, facilitating better preparation for each exam. Figure 3 shows that this problem can be modelled as a graph coloring problem, where each exam is represented by a vertex and edges join vertices that represent exams that cannot be scheduled at the same time. A solution to the graph coloring problem allocates each vertex a color such that no adjacent vertices are in the same color class and the minimum number of colors are used. This equates to a feasible timetable in the minimum number of time slots.

The next section will provide further background to the study of heuristics before we go on to consider various heuristics in the three categories mentioned previously. Hyper-heuristics and Matheuristics will then be discussed; these are two exciting variants of heuristics that have caused interest in the research community and there is hope that they have the potential to have a significant impact on solution quality and solution robustness.

Background of Heuristics

In an ideal world, there would be no need for heuristics as it would be possible to solve all optimization problems exactly. In practice, as discussed above, this is not always the case. When this happens, the user has the option of using a heuristic

method. However, another approach is possible. The user can relax attributes of the problem giving a simplified version which can be solved exactly. The solution is produced and assessed to see whether it is feasible to the original problem. If it is feasible, then it must be optimal. If not, then additional characteristics are introduced to make the relaxed problem more like the true problem, and the process continues.

For the latter method, there are difficulties. In practice, even the relaxed problems can take a long time to solve so heuristics may be used to solve them. Additionally, it is not known how many times additional characteristics will have to be added to give a feasible solution to the original problem. Therefore, research has focused on the production of high-quality heuristics.

When producing a heuristic, the following attributes should be considered:

- Heuristics should be simple and clear. Although for some problems, complicated heuristics may be required to produce solutions that are good enough, in general simple heuristics are preferable to complicated ones. Clarity is important so when describing a heuristic that others may implement, there should be no ambiguity and the heuristic should consist of well-described steps.
- Heuristics should be effective and robust. They should produce solutions of good (near-optimal) quality on instances with different attributes. For example, a good heuristic for the Travelling Salesman Problem will produce good solutions for both small and large instances, dense and sparse instances, and clustered and spread-out instances.
- Heuristics should be quick. The time they require to run should be acceptable to the user and significantly less than the run time required to find the optimal solution. The run time will depend on how the method is implemented, the resources available, etc.
- Heuristics should be flexible. In practice, problems may change with additional constraints being added or objectives changed, so the heuristic should be easily adaptable to allow for changes in the problem definition.

One key challenge is measuring the effectiveness and quality of heuristics. For many problems, benchmark datasets have been produced and are used to compare solution quality with the methods that are proposed by other researchers. Otherwise, solution quality may be compared to known bounds, or to optimal results where known. For some heuristics, a theoretical understanding of worst-case behavior is helpful. Increasingly researchers are sharing their code and results so they can be scrutinized by other researchers. This is welcome and leads to greater transparency and may lead to further collaboration between researchers leading to improved methods.

The robustness of a heuristic is also important. As stated previously, it should work effectively across a range of problem characteristics. This has been normally addressed by using benchmark instances that reflect a range of characteristics. However, there is some concern that heuristics proposed in academic literature are overly tailored to benchmark problems and may not transfer well to other problems that may be somewhat different to the benchmarks. Many heuristics are stochastic and it is important that sufficient random runs are performed that allow statistical

significance to be demonstrated when comparing the results achieved by different heuristics.

Run time is important and depends on both the hardware and the software used. There can be a tendency to use high specification computers that are available in universities but may not be available to users in industry and business. In many benchmark instances, a maximum run time is specified on a particular computer and others then need to calculate the corresponding run time on their hardware. This is a reasonable attempt to ensure fairness when comparing methods but is susceptible to some error.

Another common way of comparing heuristics is for a user to implement several heuristics and then draw conclusions as to which is best. This is a less valid way of comparing solution quality and should be avoided if possible. It is open to conscious and subconscious bias. If for example a researcher wishes to demonstrate the quality of heuristic A, and therefore implements heuristics B, C, and D for comparison, it is in their interest for heuristics B, C, and D to perform poorly. It is possible that less time will be spent on parameter optimization, or on ensuring coding is efficient and minimizes run times on these other heuristics.

We will now consider various heuristics and how they can be applied to the Travelling Salesman Problem and the Examination Timetabling Problem.

Greedy Heuristics

Greedy heuristics construct a solution from scratch, making a greedy or locally optimal decision at each step. They tend to make a choice based on what appears best in the short term, so can be of poor quality due to not looking at the longer term. For the Travelling Salesman Problem, the simplest greedy heuristic is called the nearest neighbor algorithm and consists of starting from some city and then traveling to the nearest unvisited city until all cities have been visited, at which time the salesman returns to the starting city. Johnson and McGeogh (1995) show that solution quality is within 25% of lower bounds. Slightly more complex greedy heuristics are feasible also, for example choosing the shortest edge in turn and adding it to the tour as long as this does not create a subtour. This continues until a complete tour has been generated. An insertion heuristic begins by selecting the shortest edge. It then in turn selects the city not in the subtour that has the shortest distance to any city that is within the subtour. This city is then added to the tour in the cheapest possible place and the process continues until all cities are included in the tour. Johnson and McGeogh (1995) found this method produced slightly better solutions on average than the nearest neighbor heuristic. Another option is to form the convex hull of the set of cities and use this as the starting solution. The remaining cities are then added in turn in the best possible position.

Christofides (1976) proposed a heuristic method based on finding the minimal spanning tree of the set of cities. All edges are then duplicated and the Euler cycle is constructed. The cycle is then traversed with no city being visited more than once. Christofides also proposed a method where once the minimal spanning tree has been

generated, a minimum matching on the set of nodes of odd degree is produced. This is added to the minimal spanning tree and again, the Euler cycle is produced and used to generate the tour.

Clarke and Wright (1964) produced a heuristic for the vehicle routing problem that can also be applied to the TSP. It starts from a tour constructed by alternating between the starting city and another city until all cities have been visited. The saving is then calculated for each pair of cities equal to the change in distance if we go directly between them rather than travelling via the starting city. These are then ordered in a decreasing fashion and the tour is modified accordingly, ensuring that no cycles are introduced.

Carter et al. (1996) compared a number of different greedy heuristics for the Examination Timetabling Problem of minimizing the number of timeslots required to find a feasible timetable. Exams were ordered according to different criteria and then allocated to the first feasible timeslot. The criteria were: the number of clashing exams, the total number of students sitting clashing exams, the number of students sitting each exam, and the saturation degree. Saturation degree was originally proposed by Brelaz (1979) to solve the graph coloring problem and it measures the number of feasible colors for each node (timeslots for each exam). This is a dynamic ordering as the saturation degree for each exam must be updated as the search progresses. Carter et al. (1996) create starting solutions using these greedy heuristics that are then improved by a backtracking algorithm, and they demonstrate that the dynamic saturation degree ordering produces the best results.

While greedy algorithms typically have the advantage of being quick and simple, they will often produce results of poor quality. Therefore, they may be the starting point for improvement heuristics, which are considered in the next section.

Improvement Heuristics

Also known as hill climbers or descent methods, improvement heuristics commence from some starting solution and attempt to gradually improve the quality of the solution. The starting solution may be generated randomly so for example for the Travelling Salesman Problem, a random tour can be used. For the Examination Timetabling Problem, each exam can be allocated to a random timeslot though this will almost certainly lead to an infeasible timetable. In this instance, the improvement phase of the heuristic will have to focus on producing a feasible timetable before considering the real objective. Alternatively, a modified randomized method could be used for producing a starting solution, for example for each exam, choose a timeslot at random, and allocate it as long as it does not clash with any exams already allocated to that timeslot. If there is a clash, another random timeslot is chosen and the process continues until all exams are scheduled. If no feasible allocation is found, a core sophisticated heuristic will be required.

Alternatively a greedy heuristic similar to the ones described in the previous section can be used. This has the advantage of meaning the improvement phase starts from a better solution but is likely to require a greater run time.

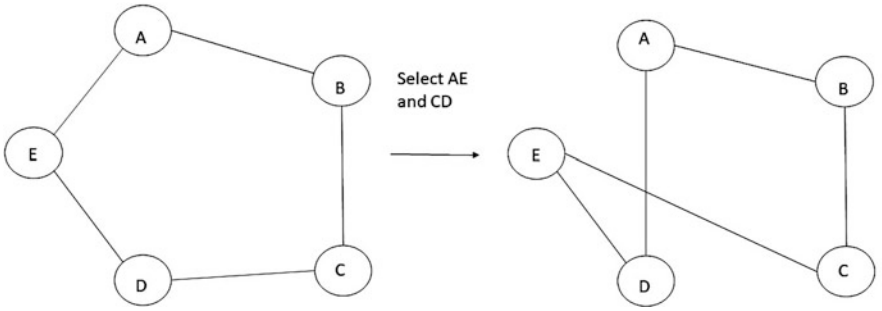

Fig. 4 An example of a 2-opt neighborhood move

The improvement phase requires a cost function *F(s)* which measures the quality of solution *s*. For the Travelling Salesman Problem, this is likely to be the total distance of the route. For the Examination Timetabling Problem, the cost function needs to measure the spread of exams for students. This may be the number of instances of students having exams in consecutive timeslots but in most research papers on this problem, the cost function that is used penalizes instances of students sitting exams within a certain number of timeslots.

An improvement heuristic also requires a neighborhood *N(s)*. This defines the set of solutions that can be reached in a single move from the current solution. For the Travelling Salesman problem, this may be constructed by deleting two edges in the tour and forming a new tour, as demonstrated in Fig. 4. Here, edges AE and CD are selected for removal and new edges AD and CE are added. This is the only possible way in which a new feasible tour can be formed. This is known as the 2-opt neighborhood.

On the related Vehicle Routing Problem many different neighborhoods have been proposed which can also be applied to the Travelling Salesman Problem. These include reversing segments of a route containing at least two customers, moving a customer or group of customers, swapping customers or groups of customers, and deleting *k* edges and forming a new tour.

For the Examination Timetabling problem, a move neighborhood would consist of the solutions obtained by moving a single exam to a new timeslot. An alternative is a swap neighborhood which exchanges two exams in the timetable though it should be noted that such a neighborhood on its own would not be able to change the number of exams allocated to each timeslot. Therefore, the swap neighborhood is normally part of a move/swap neighborhood. Other neighborhoods have been proposed, for example Dowsland and Thompson (1996) proposed a Kempe chain neighborhood consisting of a connected set of vertices formed from the subgraph induced by two colors. Consider the graph coloring shown in Fig. 5. Under a standard move neighborhood where the color of a single vertex is changed at each iteration, the only move possible is for vertex D to be colored red. Under a Kempe chain neighborhood, possible exchanges include the vertices ABEF swapping colors (the red-blue Kempe chain), AD or CF exchanging colors (the blue-green Kempe

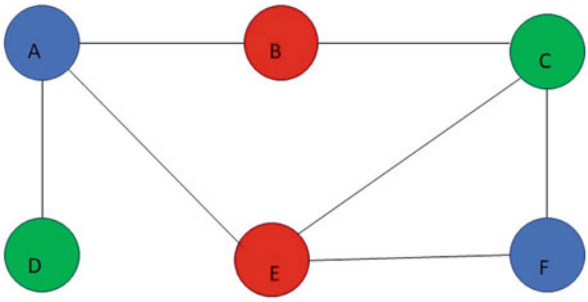

Fig. 5 An example of a Kempe chain neighborhood

chain), or BCE swapping colors (the red-green Kempe chain). This is in addition to D being re-colored red which is part of the red-green Kempe chain. Morgernstern (1989) extended this idea and coined the phrase s-chains to represent chains of moves inducted from the subgraph formed of the vertices in s colors.

We now consider various improvement heuristics in turn.

Descent Methods

Descent methods only accept neighboring solutions that improve the cost function. Assuming a minimization problem, they accept downhill moves until they become trapped in a local optimum. There are different variants of descent methods including:

- Random Descent. A random member of the neighborhood is chosen and its cost function value is evaluated. The move is accepted if the new cost function value is better than the cost function value of the previous solution, otherwise the move is rejected. This is repeated many times until it can be assumed that the search is trapped in a local optimum. This is normally approximated by measuring the number of consecutive attempted moves that are rejected and stopping when this reaches a certain value chosen by the user. This parameter is problem and instance dependent and is related to the size of the neighborhood.
- Steepest Descent. All members of the neighborhood are evaluated and the best is chosen. This is repeated until no members of the neighborhood result in an improvement in the cost function. This indicates that a local optimum has been reached and the search terminates. Compared to Random Descent, Steepest Descent has the advantage of knowing when a local optimum is reached and therefore has no parameters. However, each iteration is time-consuming as the entire neighborhood is evaluated.

These methods typically converge to poor quality solutions and are starting solution dependent, searching only a very small part of the solution space. They should only be used when an extremely quick solution is required. There have been numerous attempts to improve them, as will be considered in the following sections.

Multi-Start Methods

A single run of a descent method will be restricted to a small part of the solution space. Such a method is likely to produce a poor solution as it cannot search a wider area of the solution space. Multi-start methods repeat the descent process from different starting solutions to ensure a broader range of the solution space is searched. If starting solutions are produced randomly then the multi-start search is blind as it may re-visit parts of the solution space that have already been searched. A more intelligent multi-start search will deliberately identify parts of the solution space that have not been searched previously. For example, for the Travelling Salesman Problem, a binary matrix *Aij* records whether the search has investigated routes which include the salesman travelling from city *i* to city *j*. The next starting solution will include at least one journey *i* to *j* which has not been previously considered. For the Examination Timetabling Problem, a similar binary matrix *Bij* can be used which measures whether exam *i* has been assigned to timeslot *j*.

Simulated Annealing

Simulated Annealing is based on the study of the behavior of electrons when cooling melted metals. Metropolis et al. (1953) observed that some electrons increase in energy even though the system is cooling. This resulted in a less brittle atomic structure. Kirkpatrick et al. (1983) and Cerny (1985) built on this work and applied a method based on Simulated Annealing to large combinatorial optimization problems. Like how some atoms increase in energy, Simulated Annealing involves accepting some worsening moves, therefore enabling the search to escape from local optima and search a wider area of the solution space. The Simulated Annealing process consists of the following steps. First, a starting solution is produced, normally at random. The cost of the solution is calculated. A parameter t (temperature) is initialized and then a neighboring solution is chosen at random. If the cost of the new solution is better than or equal to the cost of the previous one, the move is accepted. If the cost of the new solution is worse than the old one, then the value of $g = \exp.(-d/t)$ is calculated where d is the increase in cost of the new solution. If a random number between 0 and 1 is less than g, then the move is accepted. Otherwise, the move is rejected. The process is repeated for a number of iterations. Then the value of t is decreased and the process continues until t is sufficiently small that it is unlikely that any further worsening moves will be accepted.

It has been shown that Simulated Annealing finds the optimal solution to a given problem (Aarts and Korst 1989). This gave Simulated Annealing academic credibility but is of little practical value as infinite time is required. Simulated Annealing does produce better solutions than descent methods due to its ability to escape from local optima. However, solution quality is very dependent on the choice of parameters, namely the starting value of t, the cooling schedule that reduces t, the

final temperature of t, and the number of iterations. These will vary between problem instances. For example, consider a Travelling Salesman Problem instance involving minimizing the distance a robotic arm must travel to place components on a circuit board. Typical increases in cost may involve a few centimeters meaning low values of t are appropriate. However if planning a route around major cities of Europe, typical increases in cost will be much greater and larger values of t are required.

The most common means of reducing the temperature t is according to $t = t * r$ where $r < 1$. The closer the value of r to 1, the slower that t reduces and therefore the longer the run takes. However, solution quality is normally improved. Mahdi et al. (2017) compare a number of cooling schedules including a logarithmic cooling schedule $t = c/\log(1 + t)$ where c is a constant, a linear cooling schedule $t = t\text{-}n$ where n is a constant, and an adaptive cooling schedule that only reduces the temperature if the cost function value is improved. The best performing was an arithmetic-geometric cooling schedule $t = at + b$ where a and b are both parameters.

The starting value of t should be chosen so that the search accepts a large number of worsening moves. To determine an appropriate value, 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 starting temperature value 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 descent method. The final temperature can be calculated by determining the smallest potential cost increase, and calculating the value of t below which there is little chance of accepting even the smallest increase in cost. The choice of the number of iterations depends on the run time available.

Malek et al. (1989) investigated the use of Simulated Annealing for the Travelling Salesman Problem. Tours are represented by arrays showing the position of each city in the tour. The neighborhood consists of solutions produced by swapping two non-adjacent edges. They use a parallel implementation with each processor running a separate annealing algorithm with different parameters. Periodically, these solutions are examined and a single promising solution is identified. Each processor then starts again from that solution, with the temperature parameter being re-initialized. Experiments were conducted with varied parameters including starting temperatures between 10 and 10,000, and cooling ratios between 0.5 and 0.98. Malek et al. (1989) found the results from the parallel implementation to be significantly better, however better results were obtained from a parallel implementation of Tabu Search.

Simulated Annealing has also been applied to the Examination Timetabling Problem. Dowsland and Thompson (1996) split the problem into two phases where the first seeks a feasible clash-free solution and the second attempts to spread the exams out for students. They observe that the solution space in the second phase may be disconnected so consider a more complex neighborhood based on Kempe chains. A standard cooling schedule was used, and it was shown that the more complex neighborhood produced superior results, though requiring additional time.

Leite et al. (2019) proposed Fast Simulated Annealing, where if an exam is not involved in a move for a certain number of iterations, it is no longer considered for moving. The same Kempe chain neighborhood was used as proposed by Thompson and Dowsland and the temperature parameter is cooled using the formula $t = t_{max}$ exp.$(-r*t)$ where r is the cooling rate and t_{max} is the starting value of t. They compare Fast Simulated Annealing with a standard Simulated Annealing implementation and achieve comparable solution quality in a reduced number of iterations.

Overall, Simulated Annealing remains an extremely popular solution method. It is relatively simple to implement and is flexible and adaptable. Although there have been attempts to automate the choice of parameters, this challenge continues to be a barrier to the method being used more widely.

Tabu Search

Both Glover (1989) and Hansen (1986) proposed Tabu Search as an alternative method of escaping from local optima. Tabu Search keeps a record of previously visited solutions and the search is not permitted to return to them. It is based on Steepest Descent meaning at each iteration, the entire neighborhood is evaluated and the best chosen. Within Tabu Search, the best non-tabu move is selected and is accepted even if it results in a worsening in the cost function value.

Recording all solutions that have been previously visited is time consuming and requires large amounts of memory. To avoid this problem, only an attribute of previously visited solutions is recorded rather than the entire solution. Additionally, only the last n moves are considered where n is a parameter known as the tabu tenure. The tabu tenure needs to be sufficiently large that the search does not merely cycle between a small number of solutions, but small enough such that the process is not too time consuming. Other than the number of iterations, the tabu tenure is the parameter in Tabu Search, which is advantageous compared with Simulated Annealing.

Recording only attributes of solutions may lead to solutions being classed as tabu even if they have not been visited before. This is a problem particularly if they are of good quality. Therefore an aspiration criterion is used which allows the search to accept a tabu move if it is of better quality than the best solution found so far. Tabu Search will head immediately to a local optimum, which is also advantageous compared with Simulated Annealing which is likely to only find good solutions toward the end of the run. Tabu Search normally starts from a reasonably good solution rather than a random one as it is a deterministic method. As well as including a short-term memory function to record tabu moves, a long-term memory function is often included to diversify the search and to guide it to unexplored regions of the solution space. Alternatively it may be used for intensification, to focus the search on areas of the solution space that are found to contain high-quality solutions. Tabu Search terminates either after a certain number of iterations or after

a given number of consecutive iterations have taken place without finding a new best solution.

He et al. (2005) use a parallel Tabu Search algorithm to solve the Travelling Salesman Problem. Each processor started with a different solution and ran independent tabu searches. After a certain time, the best solutions from each processor were reported and used to produce an initial population for a Genetic Algorithm. For the Tabu Search processes, the cost function was the tour distance and intensification and diversification strategies are included. The authors reported that a challenge is achieving an appropriate balance between the intensification and diversification phases. They achieved this dynamically, according to the progress of the search. Results are competitive with other methods.

Basu (2012) provides a summary of Tabu Search implementations for the Travelling Salesman Problem. Basu shows that a large number of neighborhood definitions have been used including removing two edges, three edges, and even four edges and creating a new solution. Vertex insertion moves a city to a new position in the route and vertex exchange swaps the positions of two cities in the route. Of the papers reviewed 26 of 47 used a static tabu tenure. Static tabu tenures vary greatly, from 5 to above 30, though most are between 5 and 16. Most of the other papers vary the tabu tenure according to problem size with larger instances requiring a larger tabu tenure. Only 1 of the 47 papers used a genuine dynamic tabu tenure that adjusted the value according to the progress of the search. Forty of the papers used an aspiration criteria and many used a frequency-based diversification strategy.

Di Gaspero and Schaerf (2001) applied Tabu Search to the Examination Timetabling Problem. They use an adaptive weighted cost function where the weights vary according to whether the corresponding objective is being satisfied or not. The neighborhood consists of all solutions that differ in the timing of a single exam. The tabu attribute records that when a certain exam moves from one time period to another, it is not permitted to change for n iterations. This is quite a restrictive neighborhood definition. They find that a greedy starting solution results in a quicker run time, and results are competitive.

White et al. (2004) also apply Tabu Search to the Examination Timetabling Problem but use a long-term memory function to record the number of times each exam is allocated to each time slot. They then avoid moving exams that have already been moved in order to diversify the search. They also employ an intensification process to intensify the search in the region containing the best solution. They find that this helps to improve solution quality. Overall, they find that Tabu Search can quickly find good solutions but struggles to identify solutions competitive to the best found in literature. The long-term memory function is easy to implement and was successful in preventing cycling and diversifying the search.

Tabu Search is a popular solution method that benefits from the lack of parameters. This does not mean that it is entirely simple to implement Tabu Search for a given problem. Most high-performing Tabu Search algorithms include a balance between intensification and diversification and much experimentation is required to achieve a suitable balance.

Threshold Acceptance

Threshold Acceptance (Dueck and Scheuer 1990), also known as Great Deluge, has similarities to Simulated Annealing, but the decision to accept a move is entirely deterministic. An initial threshold value is chosen which is greater than the cost value of the initial solution. Neighboring solutions are then chosen at random and all moves that result in a cost that is below the threshold are accepted. Gradually, the value of the threshold is reduced (for a minimization problem) meaning fewer and fewer moves are accepted. The selection of the initial threshold value is key. If it is too high, the search will accept all moves and perform a random walk. If the threshold is too low, then no moves will be accepted. It is usual for all downhill moves to be accepted regardless of whether they fall under the threshold value to ensure the search ends in a local optimum. Another key decision is how the threshold value is reduced.

Dueck and Scheuer (1990) compared Threshold Acceptance to Simulated Annealing. The cost function, starting solution, and neighborhood were the same and experiments showed that results were competitive. The method itself is simpler than Simulated Annealing and requires fewer parameters so is a heuristic deserving of more consideration. It is not as popular currently as Simulated Annealing or Tabu Search.

A variant of Threshold Acceptance named Record-to-Record was also proposed by Dueck (1993) and instead of accepting moves where the absolute objective function value lies within a certain threshold, moves are accepted if the change in objective function lies within a certain range.

Greedy Randomized Adaptive Search Procedure (GRASP)

Whereas Simulated Annealing can be considered to be an extension of Random Descent and Tabu Search can be considered to be an extension of Steepest Descent, GRASP (Feo and Resende 1995) is more similar to multi-start methods as they involve repetitions of two phases, a construction phase and an improvement phase. GRASP differs to multi-start in the way the initial solutions are generated. A construction algorithm constructs a new solution one element at a time using a selection rule that identifies the best k additions to the solution. These moves make up a restricted candidate list (RCL). Alternatively, the RCL may consist of all options that are within a certain deviation from the best. The candidate to be added to the partial solution is then randomly chosen from the RCL, thereby balancing greediness with randomness. Once a complete solution has been produced, descent is applied and the process is repeated. A variant named Reactive GRASP adjusts the size of the RCL according to the progress of the search.

Zhu and Chen (2017) select the shortest edges of a Travelling Salesman Problem to form the restricted candidate list from which a member is chosen randomly, and then use 2-opt to improve the solution, where two edges are removed and a new tour

is formed. The edges to be deleted are chosen intelligently to increase the probability of identifying improving moves. Subsequently, 3-opt is also used to improve the solution.

Casey and Thompson (2002) use the greedy heuristics proposed by Carter et al. (1996) to form the RCL and then descent with a Kempe chain neighborhood is applied to the resulting timetable. A memory function is used to diversify the search by ensuring that each newly constructed solution lies in a different part of the solution space.

Gogos et al. (2012) initially construct an exam timetable consisting of exam, time period, and room by forming an RCL of a set of examinations, which are assigned greedily to the appropriate time and room. They include a backtracking step to ensure a feasible timetable is produced. A Kempe chain neighborhood is then used to improve the solution. They incorporate a Simulated Annealing method to further improve the solution and entered their algorithm into the ITC07 timetabling competition, coming in second place.

Variable Neighborhood Search (VNS)

Variable Neighborhood Search involves creating a set of increasingly large neighborhoods for the problem at hand. VNS then converges to a local optimum using the smallest neighborhood and then attempts to improve that solution by considering a larger neighborhood. If a better solution is then found, the search reverts to the initial neighborhood; otherwise it moves to the next neighborhood. VNS takes advantage of the fact that the global optimum will be locally optimal with respect to all neighborhoods. The process begins from some starting solution x and an initial neighborhood *N1(x)*. An initial solution is generated and the search then enters a shaking phase where a point at random is chosen from the first neighborhood and local search is then applied. This is then followed by another shaking phase followed by descent with neighborhood *N2(x)*. If a better solution is found, the search returns to neighborhood *N1(x)*; otherwise the search moves to neighborhood *N3(x)* and so on.

Several researchers have attempted to improve the efficiency of VNS. As local descent can be time consuming, some omit it all together and rely on the shaking procedure alone to generate better solutions. A variant named Variable Neighborhood Descent omits the shaking phase and searches entire neighborhoods in turn. Some researchers accept some non-improving moves in order to diversify the search.

Hore et al. (2018) use a greedy algorithm to produce an initial solution to the Travelling Salesman Problem. They stress the importance of a good initial solution as this means less computational effort is required. Their first neighborhood swaps the positions of two consecutive cities in the route. Subsequent neighborhoods exchange the positions of cities two apart in the route, then three apart, etc. Local search is applied using the first neighborhood until a local optimum is reached. The neighborhood is then extended to also include the members of the neighborhood

N2. This continues until a stopping condition is satisfied. If no improving solution is found, then a previously visited tour is selected randomly and the search begins again.

Rather than choosing a random shake move, Burke et al. (2001) identify edges included in the tour that are expensive relative to the minimum edges from the endpoints. Their shaking process consists of removing three edges and forming a new tour randomly. One of these edges is chosen from the set of expensive edges. The number of edges in the shaking process is gradually increased, corresponding to more complex neighborhoods. They show that their results are better than those obtained under a standard local search method.

Burke et al. (2010) solved the Examination Timetabling Problem using nine neighborhoods within a VNS framework. These include moving a single exam, swapping two exams, moving two exams, a Kempe chain move, and moving entire timeslots. They do not just accept improving moves but also those which are less than 1% worse than the current solution. They also bias the shake move by selecting exams that are contributing to the cost function They also implement a Genetic Algorithm to help to identify the neighborhoods that are actually contributing to improvements in solution quality. By excluding poorly performing neighborhoods, efficiencies can be made. The method performs well across all datasets and it is interesting to note that the Genetic Algorithm found that different neighborhoods were of good quality on different problem instances.

Ayob et al. (2006) implement a descent-ascent Variable Neighborhood Search to solve the Examination Timetabling Problem. They use four neighborhoods consisting of a swap neighborhood, a move neighborhood, a Steepest Descent swap neighborhood, and a stochastic neighborhood that probabilistically accepts worse solutions with the probability dependent on solution quality, run time, and whether the search appears to be trapped in a local optimum. The method was competitive on the benchmark instances.

Population-Based Methods

In this section, several methods that use a population of solutions are described.

Genetic Algorithms

Genetic algorithms (GAs) were proposed by Holland (1992) as a method for solving optimization problems based on some of the processes observed in natural and biological evolution. Genetic Algorithms are based on the idea of survival of the fittest and combine attributes from different solutions to form new ones. A GA begins with a population of solutions to the given problem. They are subject to genetic operators, namely crossover that combines two parent solutions and forms child solutions, and mutation which introduces random variation and avoids the search converging to a set of identical or similar solutions.

An important decision when implementing a GA for a given problem is the problem representation. This may be binary, integer, permutation, continuous, etc. For the Travelling Salesman Problem, a permutation representation is common whereas for the Examination Timetabling Problem, an integer representation is more common.

A GA consists of the following main features:

- An initial population. These are typically generated at random but may be produced using a number of randomized heuristics. It is important the population is sufficiently diverse, so some researchers deliberately ensure that the set of initial solutions contain significant differences. The size of the population is a parameter. If the population size is too small, then there will be insufficient diversity and the search may quickly converge. If the population is too large, then the search may lack focus and be time consuming.
- Parent selection. From the population, two parent solutions need to be selected. This may be done randomly or using tournament selection, where n solutions are chosen at random and the best of these n is chosen to be a parent. Alternatively, probabilistic selection may be used where the probability of selecting a certain solution is proportional to the quality of that solution. This is known as roulette wheel selection. For minimization problems such as the Traveling Salesman Problem and the Examination Timetabling Problem, the probability of selection is proportional to the reciprocal of the cost function value.
- Crossover. This is the process by which parent solutions are combined and new child solutions are formed. They need to ensure that the child solutions inherit attributes of the parent solutions. For the Examination Timetabling problem with an integer representation, possible crossover operators include 1-point, 2-point, and uniform. Under 1-point crossover, a random cut point is chosen and the first child solution is made up of the first parent solution before the crossover point combined with the second parent solution after the crossover point. Under 2-point crossover, two random cut points are selected and the child solution is made up of the elements before the first crossover point and after the second crossover point from the first parent, and the elements between the two cut points from the second parent. Uniform crossover selects each element randomly from each parent. This is illustrated in Fig. 6. For problems like the Travelling Salesman Problem with a permutation representation, these crossover operators are not appropriate as they would leave to infeasible children due to missing and repeated values. Therefore, alternative crossovers such as partially mapped crossover and order crossover are appropriate. For partially mapped crossover, two cut points are chosen at random and a mapping system is defined that ensures offspring are legitimate. For order crossover, again two random cut points are chosen. The child consists of the elements from the parent from between the two points, with the remaining values being added in the order they occur in the other parent. These crossovers are illustrated in Fig. 7.

Fig. 6 Example of uniform crossover

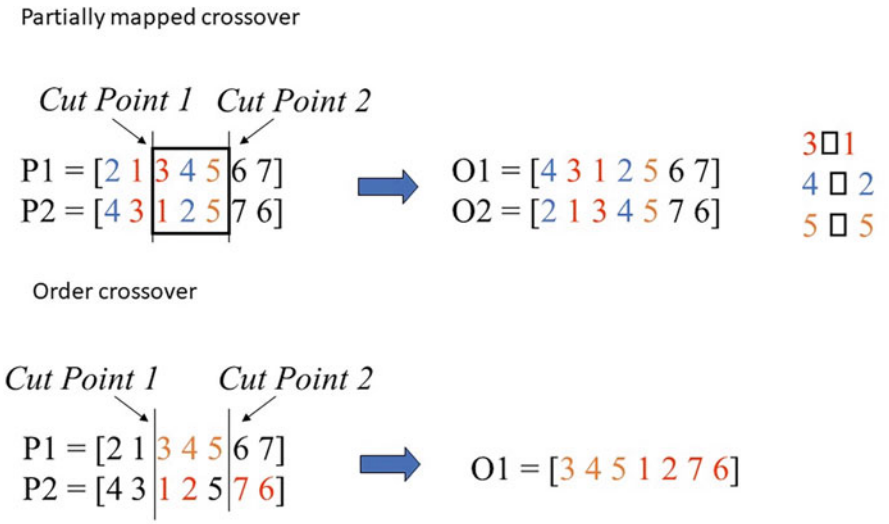

Fig. 7 Examples of partially mapped crossover and order crossover

- Mutation. These operators add random variation, ensuring diversity in the population. A mutation parameter between 0 and 1 is defined and for each element in the child solution, a random number is generated. If it is less than the mutation parameter, then mutation is applied. For integer representations such as the Examination Timetabling Problem, these can just be the selection of a new time slot for the selected exam. More intelligent mutations may involve only permitting the move if the new timeslot is feasible or performing a search to identify the best slot for this exam. For the TSP, a typical mutation operator would exchange the position of two cities in the route.
- Population Update. The population size normally stays the same throughout a run so as new child solutions are produced, other solutions must be removed from the population. This may involve the worst member of the population being removed, a probabilistic choice or an entirely new generation of solutions being generated. In the latter case, elitism is often included so the best solutions from one generation remain in the next.
- Stopping criteria. This may depend on run time or a fixed number of generations. A common issue is that over time, diversity within the population diminishes and the population consists of similar solutions. This can be monitored and changes introduced, for example entirely new solutions can be

added to the population or solutions with attributes that differ from the solutions in the population can be generated.

An issue for Genetic Algorithms is they can be very slow. Therefore, many Genetic Algorithms include a descent method in order to find good solutions quicker, though applying descent itself takes time. In a Memetic Algorithm, descent is applied to all members of the initial population and to all child solutions, meaning the population always consists of solutions that are local optima with respect to the neighborhood used.

Ahmed (2010) uses a permutation representation for the TSP, thereby ensuring that all solutions are feasible. Roulette wheel selection is used and a new Sequential Constructive Crossover Operator is introduced. From a given node, this operator considers the next city to visit based on the two options in the parent solutions and selects the shortest path. Ahmed demonstrates that this operator produces better solutions than either an edge recombination crossover or an N-point crossover.

Wang (2014) produces the initial population of routes randomly, and a form of roulette wheel selection is used to choose parents. 2-point crossover is used as well as a swap mutation operator. Local optimization strategies are used to enhance the quality of the child solutions. The author emphasizes the importance of the selection of appropriate parameters.

Dener and Calp (2018) use an integer representation for their Genetic Algorithm for the Examination Timetabling Problem. A course assignment algorithm is used to generate the initial population. As it considers exams in a random order, there is some diversity in the initial population. Roulette wheel selection is used to choose parents and a multi-point crossover operator is used. A swap mutation operator is used and a repair operator ensures that all children are feasible.

Jha (2014) uses a large binary representation to include information about the room and invigilator as well as the time period of each exam. A random initial population is generated and roulette wheel selection is used to produce parent solutions. 1-point crossover is used. Random changes are made via a mutation operator and the author reports that the Genetic Algorithm is a powerful method for solving the timetabling problem.

Memetic algorithms are a very popular solution method for a wide variety of combinatorial optimization problems. They can require long run times and require the selection of a number of parameters including the population size and the mutation rate. They are an efficient method for producing good solutions to many problems.

Ant Colony Optimization (ACO)

Ants have been observed to find the shortest path between a food source and their nest. This is achieved via the ants creating paths and leaving a trail of pheromone which subsequently influences the decisions of other ants as to which paths to follow. This has a cumulative effect meaning that the trail is re-enforced on shorter

paths. More and more ants follow this shorter path due to the stronger trail until the trail is sufficiently strong that all ants follow the shortest path. Information about this can be found in Dancubourg et al. (1990).

Dorigo et al. (2006) used models of the real-life behavior of ants to solve combinatorial optimization problems. A series of agents (ants) construct solutions using probabilistic greedy algorithms. The probabilities have two components; a greedy element that measures the short-term cost of that decision; and a trail component that measures the quality of previous solutions that have included that decision.

Dorigo et al. (2006) applied a number of ACO variants to the TSP. At each iteration of the greedy algorithm, the decision as to which city j should be visited from current city i is calculated according to the equation:

$$p_{ij} = \frac{\tau_{ij}^{\alpha}.\sigma_{ij}^{\beta}}{\sum \tau_{ij}^{\alpha}.\sigma_{ij}^{\beta}}$$

where τ is the trail value indicating the quality of previous solutions that have included path i-j, and σ is the reciprocal of the distance, meaning the probability of selection is higher for closer cities. A and β control the relative importance of the trail versus the heuristic information. When the first set of routes have been constructed, the trail values are updated according to the equation:

$$\tau_{ij} = (1 - \rho)\, \tau_{ij} + \sum Q/L_k$$

where ρ is the evaporation rate, Q is a constant, and L_k is the length of the tour constructed by ant k. The value Q/L_k is added to each route ij that is taken by ant k.

Variants of the initial ant system include ant colony system which includes a local pheromone trail update step $\tau_{ij} = (1 - \rho)\tau_{ij} + \rho\tau_0$, where t_0 is the initial trail value. The trail is updated after each construction step and the local update is introduced to diversify the search by reducing the trail on the routes that have been traversed, encouraging the use of less used routes. Another variant is the max-min which only uses the best ant to update the trail and ensures all trail values lie within a certain range. Another variant is due to Bullnheimer (1999) who ranks the ants according to their tour length and use these rankings to weight the pheromone added by each ant.

Hlaing et al. (2011) attempt to improve the ant colony optimization algorithm by introducing a dynamic candidate list that limits the number of cities that are considered for adding to the route at any point of time to the closest k, where k is a parameter and is dependent on the size of the problem. They also vary the value of beta dynamically during the search to reflect the fact that as more trail information is added, less emphasis needs to be placed on the heuristic data.

Ant colony was applied to the Examination Timetabling Problem by Dowsland and Thompson (2005) who use a graph coloring model to produce timetables in the minimum number of timeslots. The greedy heuristic uses variants of the

well-known graph coloring RLF and Dsatur algorithms. The trail array *(i, j)* measures the quality of solutions when exams *i* and *j* are allocated to the same timeslot. When deciding which timeslot to allocate an exam to, the average trail with all exams already allocated to that timeslot is considered. The authors introduce an improved fitness function that measures how far each solution is from a solution with fewer timeslots, and a diversification function.

Ant colony optimization is a relatively complex algorithm that also requires a number of parameters to be determined. However, it has demonstrated the ability to perform well on a number of different problems.

Bee Colony Optimization

The Bee Colony Optimization Algorithm is based on the behavior of bees when they search for food. Each bee finds sources of nectar by following other bees that have already discovered food sources. Once the bee retrieves some nectar and delivers it to the hive, it can either abandon that food source and look for a new one, return to the same food source, or recruit new bees to follow it to this source.

Karaboga and Basturk (2007) proposed an artificial Bee Colony Algorithm where bees are either employed, onlookers, and scouts. The algorithm begins by assigning all employed bees to a randomly generated solution, which corresponds to a food source. At each iteration, they search the neighborhood and evaluate their fitness (nectar). If this is an improved solution, then the bee moves there. When all the bees have finished this process, they share this information with onlooker bees who each selects a food source according to the fitness function values. After all onlookers have selected their food sources, each of them determines a food source in its neighborhood and computes its fitness. If an onlooker finds a better fitness value, it changes the food source. 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.

Khan and Maiti (2019) employed bees to generate random tours which are then improved using local descent with swap neighborhoods. Subsequently, each onlooker bee selects a solution according to their fitness and attempts to improve it further. New solutions are then generated by scout bees and the process continues.

In Wong et al. (2008), bees construct complete tours using a combination of the arc distance and a preferred path which has been passed on by another bee that has already constructed a solution. They do not use a local search phase but rely on the intelligent construction of new solutions and the passing of information from one set of bees to the next to enhance solution quality. Competitive results are produced.

Alzaqebah and Abdullah (2011) construct an initial solution to the Examination Timetabling problem using the largest degree first heuristic. Employed bees then construct random solutions and improve them with a neighborhood descent

approach using one of ten neighborhoods selected at random. Onlooker bees then focus on the better solutions and enhance them with further neighborhood structures. Finally scout bees determine which solutions should be abandoned and replace them with new food sources. Competitive results on standard benchmark datasets are produced. Boloji et al. (2012) consider the effect of the neighborhood structures and find that including swap, move, and Kempe chain neighborhoods is sufficient for bee colony to identify good solutions.

Bee Colony Optimization has demonstrated considerable potential on a number of problems but is not easy to get working well and has yet to be broadly adopted by the optimization community. It would be good to see further research on this algorithm to see whether it can compete across a broad spectrum of problems with more established algorithms.

Harmony Search

Harmony Search was proposed by Geem et al. (2001) and is based on how musicians improvise when searching for a pleasing harmony. A harmony matrix is initialized consisting of random solutions. New solutions are then generated based on randomly selecting values of solutions in the harmony matrix or other random values. Solutions are then subject to some local variation. Alia and Mandava (2011) point out that they produce offspring based on all solutions in the population (harmony matrix) which is advantageous over Genetic Algorithms which only consider two parent solutions. They also point out that the method is reliant on several parameters such as the size of the harmony matrix, the probability of selecting information from the harmony matrix and parameter selection is of vital importance and can be difficult so varying them in a dynamic fashion can be helpful. Several researchers have hybridized Harmony Search with other meta-heuristics to improve solution quality.

Antosiewicz et al. (2013) apply Harmony Search to the Travelling Salesman Problem and compare the results with five other meta-heuristics. A run time of 100 seconds is used for each method and they find that Simulated Annealing produces the best results. In general, the population-based methods perform poorly which is probably due to the limited run time. Harmony Search evaluates fewer solutions than all the other methods considered including a Genetic Algorithm and Particle Swarm.

Al-Betar et al. (2010) use random solutions produced using a saturation degree heuristic to produce the initial harmony matrix for the Examination Timetabling Problem. In the next step, new solutions are produced by considering the exams in the order of fewest feasible timeslots. The timeslots each exam is assigned will depend either on the solutions in the harmony matrix or randomness. Each exam then has a small chance of being re-assigned using either a move, swap, or Kempe chain change. Throughout, feasibility is maintained. The new solutions replace the worst members of the harmony matrix.

Other Heuristic Methods

Even though we have considered a number of different meta-heuristic methods, there are still more that could have been included. Iterated Local Search oscillates between a construction phase which diversifies the search and a local search phase and Particle Swarm Optimization uses a model of how birds flock to solve optimization problems. It creates a population of particles that move around the search space, guided by the best-found solutions.

Meanwhile researchers continue to propose and develop new general heuristic methods. However, none has yet become as popular as Simulated Annealing and Genetic Algorithm, for example. Researchers are also continually making improvements to the implementation of these heuristic methods. In the next section, we consider two general heuristic methods that have the potential to compete with these well-established methods and to produce excellent quality results.

Matheuristics

Advances in computational power have led to increases in the size of problem that can be solved exactly in a reasonable amount of time. Although they are still not capable of solving many large problems quickly, some researchers have looked to combine exact approaches with heuristic methods. Such hybrid methods are known as Matheuristics. Archetti and Garcia Speranza (2014) classify them into the following categories.

Decomposition Approaches

The essence of decomposition approaches is based on the concept of dividing the problem into subproblems that are easier to solve. For the Travelling Salesman Problem, an example would be to divide the map of cities into regions that are sufficiently small that the optimal route of the cities within each region can be found exactly. These solutions are then merged into a combined tour. For the vehicle routing problem, a cluster first route second approach firstly assigns customers to vehicles and then finds the route for each vehicle.

Rolling horizon approaches apply to problems where decisions must be made over time. The time period is divided into intervals and each subproblem is solved in turn, with decisions made in one subproblem being fixed for the following subproblem.

Improvement Heuristics

Improvement heuristics may include Matheuristics where a heuristic is used to produce a starting solution for an exact method, or where an exact method is used to generate a solution to a relaxed problem that is then the starting point for a subsequent heuristic. Exact methods can also be used to search large neighborhoods

within a heuristic method. Some researchers have used exact methods to intensify the search at various points within a heuristic.

Relaxations to Exact Approaches

Archetti and Speranza refer specifically to Branch and Price and column generations methods, but this category can be considered more broadly as relaxations to exact approaches. For example, branch and bound methods can be speeded up by using heuristics to prune branches which are unlikely to contain better solutions. There is a risk that the optimal solution may be excluded. Another approach is to use heuristic methods to generate high-quality bounds in order to prune the search tree. Alternatively, a relaxed version of the problem may be solved exactly and this solution is then used as a starting point for a heuristic approach. In some column generation approaches a heuristic may be used to create a feasible solution quickly and branching decisions may be made heuristically in order to exclude more solutions from consideration.

Examples of Matheuristics applied to the Travelling Salesman Problems and Examination Timetabling Problems are rare. Masoud et al. (2020) solve a variant of the Travelling Salesman Problem where a salesperson collects rewards from activities over a number of days. A two-phase method is used. The Matheuristic first decomposes the cities into a number of subgroups equal to the number of days. The subproblem for each day is then solved exactly using an integer program.

Bargetto et al. (2016) apply a large neighborhood search with multiple neighborhoods to solve the Examination Timetabling Problem. One neighborhood selects a time window consisting of a number of consecutive days and the subproblem of optimizing the exams currently scheduled in that time window is solved exactly.

Hyper-Heuristics

Hyper-heuristics can be considered to be high-level approaches that can select from a number of low-level heuristics at each decision point. Their aim is to be more generally applicable across problem instances and even different problems, by only using limited problem-specific information. Burke et al. (2010) classified Hyper-heuristics according to whether they choose from a set of existing heuristics or whether they generate new heuristics. Some are perturbative and only consider complete solutions, and some are constructive and consider partial solutions. Most Hyper-heuristics use some form of learning to guide the search according to information found in previous iterations.

Selection Methods Between Constructive Heuristics

These approaches gradually construct a solution using a set of constructive heuristics to gradually build a solution. The Hyper-heuristic will attempt to identify the heuristic that is most appropriate to make the next addition to the solution. This continues until a complete solution is formed. Information from this construction can then be fed back to the following iterations.

Selection Methods Based on Perturbative Heuristics

These generate a complete solution and then select and apply a chosen improvement heuristic. The selection of the heuristic may be offline, so the choices are all known in advance or online meaning they depend on previous decisions. The latter is performed by generating scores for each heuristic based on their performance and incorporating this information in later choices. The heuristic may be chosen according to roulette wheel selection or some other system. The decision still needs to be made as to whether to accept the generated heuristic perturbations. Possibilities include accepting all moves or only improving moves, or a method such as Simulated Annealing or Tabu Search can be applied.

There are several examples of Hyper-heuristics being applied to the Examination Timetabling Problem. Burke et al. (2006) maintain a case base of example problems and information as to the best performing heuristics on them. Further problems are then solved by identifying the problems in the case base which are most similar and applying the corresponding heuristics that worked well on the problem in the case base. Yang and Petrovic (2004) use a similar approach but apply the Great Deluge Algorithm to improve the solution. Burke et al. (2007) use Tabu Search to improve a solution constructed using heuristics including largest degree, largest weighted degree, the number of students, and random.

Bilgin et al. (2006) compare different heuristic selection methods including Random Descent, Tabu Search, and a greedy method and move acceptance methods based on accepting all moves, improving moves only, and great deluge. Burke et al. (2008) use Simulated Annealing for move acceptance within a Hyper-heuristic.

Interest in Hyper-heuristics appears to have lost momentum in recent years. However, the search for methods that can adapt to different problems and instances remains and as Hyper-heuristics are particularly suited to this, their potential should not be underestimated.

Conclusions

This chapter has provided an overview of the exciting research area of heuristics. It is clear that interest in heuristics remains high. Many different heuristic methods have been described and these have different advantages and disadvantages. Ultimately, the decision of which to use for a particular application often comes down to the preference of the programmer. The quality of the implementation is crucial, so a good implementation of a poor heuristic will normally perform better than a bad implementation of a good heuristic. Each heuristic can be applied in different ways, and much thought should be given as to which is the best implementation for a particular problem. It is sensible to review the academic literature to learn from the work of others what might work well.

Many heuristics depend on the choice of parameters and further research is needed on dynamic parameter selection, which is capable of selecting the best parameters for a particular instance. Adaptive search methods that dynamically learn and make use of the information to guide the heuristic is also an interesting area

of research. Hybridizing heuristics to form stronger methods is also an interesting research area and the interested reader is referred to Salhi and Thompson (2022).

References

Aarts E, Korst J (1989) Simulated annealing and Boltzmann machines, Interscience series in discrete mathematics and optimisation. Wiley

Ahmed Z (2010) Genetic algorithms for the travelling salesman problem using sequential constructive crossover operator. Int J Biom Bioinforma 3:6

Al-Betar M, Khader A, Thomas J (2010) A combination of metaheuristic components based on search for the uncapacitated examination timetabling problem. 8th international conference on the practice and theory of automated timetabling, Belfast, pp 57–80

Alia O, Mandava R (2011) The variants of the harmony search algorithm: an overview. Artif Intell Rev 36:49–68

Alzaqebah M, Abdullah S (2011) Artificial bee colony search algorithm for examination timetabling problems. Int J Phys Sci 6(17):4264–4272

Antosiewicz, A, Koloch, G, B Kaminski B (2013) Choice of best possible metaheuristic algorithm for the travelling salesman problem with limited computational time: quality, uncertainty and speed. J Theor Appl Comput Sci 7(1):46–55

Archetti C, Garcia Speranza M (2014) A survey on matheuristics for routing problems. EURO J Comput Optim 2(4):223–246

Ayob M, Burke E, Kendall G (2006) An iterative re-start variabe neighbourhood search for the examination timetabling problem. Practice and theory of automated timetabling, Brno, Czech Republic, pp 336–344

Bargetto R, Della Croce F, Salassa F (2016) A matheuristic approach for the examination timetabling problem. 11th international conference on the practice and theory of automated timetabling, Udine, Italy, pp 467–471

Basu S (2012) Tabu search implementation on travelling salesman problem and its variations: a literature survey. Am J Oper Res 2(2):163–173

Bilgin B, Ozcan E, Korkmax E (2006) An experimental study on hyper-heuristics and exam timetabling. The third international conference on the practice and theory of automated timetabling, Brno, Czech Republic, pp 123–140

Boloji A, Khader A, Al-Betar M, Awadallah M, Thomas J (2012) The effect of neighbourhood structures on examination timetabling with artificial bee colony. 9th international conference on the practice and theory of automated timetabling, Son, Norway, pp 131–144

Brelaz D (1979) New methods to color the vertices of a graph. Commun Assoc Comput Machin 22:251–256

Bullnheimer B (1999) A new rank based version of the ant system: a computational study. Cent Eur J Oper Res 7:25–38

Burke E, Cowling P, Keuthen P (2001) Effective local and guided variable neighbourhood search methods for the asymmetric travelling salesman problem. Workshops on Applications of Evolutionary Computation, Berlin, pp 203–212

Burke E, Petrovic S, Qu R (2006) Case based heuristic selection for examination timetabling. J Sched 9:115–132

Burke E, McCollum B, Meisels A, Petrovic S, Qu R (2007) A graph based hyper-heuristic for educational timetabling problems. Eur J Oper Res 176:177–192

Burke E, Kendall G, Misir M, Ozcan E (2008) A study of simulated annealing hyper-heuristics. In: Proceedings of the international conference on the practice and theory of automated timetabling

Burke E, Eckersley A, McCollum B, Petrovic S, Qu R (2010) Hybrid variable neighbourhood approaches to university exam timetabling. Eur J Ops Res 206:46–53

Carter M, Laporte G, Lee S (1996) Examination timetabling: algorithmic strategies and applications. J Oper Res Soc 47(3):373–383

Casey S, Thompson J (2002) GRASPing the examination scheduling problem. Practice and theory of automated timetabling IV, Lecture notes in computer science 2740:232–246
Cerny V (1985) Thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. J Optim Theory Appl 45:41–51
Christofides N (1976) Worst case analysis of a new heuristic for the travelling salesman problem, Report No. 338. Carnegie-Mellen University, Pittsburgh
Clarke G, Wright J (1964) Scheduling of vehicles from a central depot to a number of delivery points. Oper Res 12:568–581
Dancubourg J, Aron A, Goss S, Pasteels J (1990) The self organising exploratory pattern of the Argentine ant. J Insect Behav 3:159–168
Dener M, Calp M (2018) Solving the exam scheduling problem in central exams with genetic algorithms. Muğla J Sci Technol 4:102–115
Di Gaspero L, Schaerf A (2001) Tabu search techniques for examination timetabling. In: Practice and theory of automated timetabling III, Konstanz, Germany, Selected Papers, pp 104–117
Dorigo M, Biratari M, Stutzle T (2006) Ant colony optimisation. IEEE Comput Intelli 1:28–39
Dowsland K, Thompson J (1996) Variants of simulated annealing for the examination timetabling problem. Ann Oper Res 63:105–128
Dowsland K, Thompson J (2005) Ant Colony optimisation for the examination scheduling problem. J Oper Res Soc 46:426–438
Dueck G (1993) New optimization heuristics: the great deluge algorithm and the record-to-record travel. J Comput Phys 104(1):86–92
Dueck G, Scheuer T (1990) Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. J Comput Phys 90(1):161–175
Feo T, Resende M (1995) Greedy randomized adaptive search procedures. J Glob Optim 6:109–133
Geem Z, Kim J, Loganathan J (2001) A new heuristic optimization algorithm: harmony search. Simulation 76:60–68
Glover F (1989) Tabu search – part 1. ORSA J Comput 1(3):190–206
Gogos C, Alefragis P, Housos E (2012) An improved multi-staged algorithmic process for the solution of the examination timetabling problem. Ann Oper Res 194:203–221
Hansen, P (1986) The steepest ascent mildest descent heuristic for combinatorial programming. Presented at the congress on numerical methods in combinatorial optimisation, Capri, Italy
He Y, Qiu Y, Liu G, Lei K (2005) A parallel adaptive Tabu search approach for travelling salesman problems. International conference on natural language processing and knowledge engineering, IEEE, Jeju Island, Korea, pp 796–801
Hlaing Z, Khine M (2011) Solving the traveling salesman problem by using improved ant colony optimisation algorithm. Int J Inf Educ Technol 1(5):404
Holland J (1992) Genetic algorithms. Sci Am 267(1):66–73
Hore S, Chatterjee A, Dewanji A (2018) Improving variable neighbourhood search for the traveling salesman problem. Appl Soft Comput 68:83–91
Jha S (2014) Exam timetabling using genetic algorithms. Int J Res J Eng Technol 3(5):649–654
Johnson D, McGeogh L (1995) The travelling salesman problem: a case study in local optimisation. Local search in combinatorial optimisation 1(1):215–310
Karaboga D, Basturk B (2007) Artificial bee colony optimization algorithm for solving constrained optimisaton problems. In: Melin P, Castillo O, Aguilar L, Kacprzyk J, Pedrycz W (eds) Foundations of fuzzy logic and soft computing. Lecture notes in lecture science 4529. Heidelberg, Berlin
Khan I, Maiti M (2019) A swap sequence based artificial bee colony algorithm for the travelling salesman problem. Swarm Evol Comput 44:428–438
Kirkpatrick S, Gelatt C Jr, Vecchi M (1983) Optimisation by simulated annealing. Science 220:671–680
Leite N, Melicio F, Rosa A (2019) A fast simulated annealing algorithm for the examination timetabling problem. Expert Syst Appl 122:137–151

Mahdi W, Medjahed S, Ouali M (2017) Performance analysis of simulated annealing cooling schedules in the context of dense image matching. Comput Sist 21(3):493–501
Malek M, Guruswamy M, Pandya M, Owens H (1989) Series and parallel simulated annealing and Tabu search algorithms for the travelling salesman problem. Ann Oper Res 21:59–84
Masoud S, Aksen D, Salhi S (2020) Formulation and a 2-phase matheuristic for the roaming salesman problem: applications to election logistics. Eur J Oper Res 280(2):656–670
Metropolis N, Rosenbluth A, Rosenbluth M, Teller A, Teller E (1953) Equation of state calculations by fast computing machines. J Chem Phys 21:1087–1092
Morgernstern C (1989) Algorithms for general graph coloring. Doctoral dissertation. Department of Computer Science, University of New Mexico
Salhi S, Thompson J (2022) The new era of hybridization and learning in heuristic search design. In: The Palgrave handbook of operational research, pp 501–538
Wang Y (2014) The hybrid genetic algorithm with two local optimization strategies for the travelling salesman problem. Comput Ind Eng 70:124–133
White G, Xie B, Zonjic S (2004) Using Tabu search with longer term memory and relaxation to create examination timetables. Eur J Oper Res 153:80–91
Wong L, Low M, Chong C (2008) A bee colony optimization algorithm for the travelling salesman problem. Second Asia international conference on modelling and simulation, Kuala Lumpar, Malaysia, pp 818–823
Yang Y, Petrovic S (2004) A novel similarity measure for heuristic selection in examination timetabling. Third international conference on the practice and theory of automated timetabling, Konstanz, Germany, pp 247–269
Zhu M, Chen J (2017) Computational comparison of GRASP and DCTSP methods for the travelling salesman problem. 2nd international conference on image, vision and computing, IEEE, Chengdu, China, pp 1044–1048