Optimisation With Hillclimbing On Steroids: An Overview of Neighbourhood Search Techniques

Andrew Tuson
Department of Artificial Intelligence, University of Edinburgh,
5 Forrest Hill, Edinburgh EH1 2QL, Scotland, U.K.
Email: andrewt@dai.ed.ac.uk

Abstract

This paper gives an overview of a class of optimisation techniques commonly known as 'neighbourhood search', of which examples include simulated annealing, tabu search and evolutionary algorithms. This overview will be followed by a discussion of an approach to the design of these techniques, and how domain knowledge can be exploited. References for further reading will be provided.

Introduction

Consider a **Combinatorial Optimisation Problem** (COP) which has the form:

- A space of possible solutions, S, with possibly constraints on valid solutions;
- A 'quality' for each solution, $quality(s)$ where $s \in S$;

where the objective is to find a solution s such that $quality(s)$ is maximised and the constraints satisfied. Such problems are usually NP-hard[1] and therefore, for a problem of large enough size, the time taken to find an optimal solution by exact methods will become prohibitive. Fortunately, in practice a 'good enough' answer in the time available is all that is required, which opens up the possibility of using **heuristic** methods that do not guarantee optimality, but work well in practice. One heuristic approach would be to employ some form of **hillclimbing**, which requires the following decisions to be made:

1. Find a suitable **encoding scheme** for the candidate solutions in S.
2. A **quality measure** for each solution (ie. $quality(s)$).
3. A method of modifying an encoded solution to give another solution (a **move operator**).

There is usually more than one possible solution (valid or invalid) that applying the move operator on a given solution can produce; therefore we define a **neighbourhood**, $N(s, m)$, as the set of solutions in S that are produced by applying the move operator, m, on a solution s. Once the above have been defined, the general algorithm is quite straightforward and is described below:

1. Generate an initial solution (usually at random) and evaluate it;
2. Apply and evaluate a move in the neighbourhood of the current solution and apply an **acceptance criterion** to decide whether to use the new solution;
3. Go back to step 2 until a **termination criterion** is reached.

The termination condition is merely when the user would like to stop the search, examples would include when a certain amount of CPU time has elapsed, or when a solution of a certain quality has been found. The acceptance criterion determines whether a new solution generated by a move operator replaces the current solution, and introduces a bias towards better solutions in the search. Acceptance criteria, and thus hillclimbers, can be broadly classified as follows.

- **Any-Ascent** hillclimbers (AHC) accept moves that give solutions, s_{new}, that have better or equal quality than the current solution, s_{curr} (ie. accept when $quality(s_{new}) \geq quality(s_{curr})$). A common variant is **stochastic** hillclimbing (SHC) where moves are tried in a random order.
- **First-ascent** (FAHC) hillclimbers, operate similarly, but take the first *improvement* in quality found (ie. accept if $quality(s_{new}) > quality(s_{curr})$).
- **Steepest-ascent** (SAHC) hillclimbers, systematically evaluate the entire neighbourhood of the current solution and accept the most improving move (ie. accept if $quality(s_{best}) \geq quality(s_{curr})$ and $quality(s_{best}) \geq quality(s_{any})$ $(s_{any} \in N(s_{curr}, m))$).

The bias introduced by the acceptance criterion, though necessary, can lead to problems an implicit assumption is made that the **landscape**[2], which is a graph induced by the move operator and quality function that connects solutions accessible to each other by a move operator, is correlated in such a way as to lead the hillclimber into a region of the landscape with high-quality solutions. Deviations from this ideal can lead to the hillclimber being **deceived** (the correlations lead the hillclimber away from the optimum), or stuck in **local optima**. Fortunately, these potential difficulties do not necessarily arise in practice, as this assumption is not too far from the reality if a suitable move operator is used.

Extending the Hillclimber. A recent class of combinatorial optimisation techniques often termed **meta-heuristics** have been developed to tackle difficult COPs.

They are distinct from conventional heuristics in that they are not tied to a single problem, but are in fact *templates* from which problem-specific optimisers can be derived. The most common type of meta-heuristic, commonly known as **neighbourhood (or local) search** extends hillclimbing in some fashion, usually by relaxing the acceptance criterion, so to escape local optima, and thus they can be placed in a common implementational framework such as described by the pseudo-code below[3]:

```
neighbourhood_search()
{
   initialise(P);             // generate starting solution(s)
   while(!finished(P)){
      Q = select_solution(P); // choose solution(s) to perturb
      R = create_moves(Q);    // apply move operator(s)
      P = merge_sets(P,Q,R);  // merge to obtain a new set P
  }
}
```

where P, Q, and R are sets of solutions in S, although it is usual to have only one solution in a set — **evolutionary algorithms**, described later, are the exception to this rule. With the problem encoding, move operator(s), quality measure specified, and by defining the above functions appropriately, any of the versions of neighbourhood search described in the remainder of this paper can be implemented.

This review paper will provide an overview of the various neighbourhood search techniques and give references for further reading (though the reader is directed to other introductions[4,5,6,7]). Finally, an approach for the design of, and these techniques will be outlined and discussed, and ways of adding domain knowledge briefly reviewed.

Iterated Hillclimbers and GRASP

One way out of the problem of local optimal is to restart the hillclimber with a different initial solution when a local optimum has, or suspected to have, been found — this is known as **iterated hillclimbing**. A common criterion for restart is when a certain user-defined number of evaluations have been made without an improvement in solution quality. the search will then resume in a different part of the search space, with a different, and possibly better quality, local optimum.

A variant of this, **GRASP** (Greedy Randomised Adaptive Search Procedure[8], a trade mark of Optimization Alternatives, Austin, Texas) incorporates a **construction phase** where the new starting point is generated using a greedy algorithm which is then followed by a local search **improvement phase**. The intelligent **initialisation procedure** used in GRASP attempts to start the search in the vicinity of good solutions. More emphasis is thus placed on the initialisation procedure than the other components of neighbourhood search, to the extent that its design is highly problem-specific and it often is *adaptive* in nature, making use of past experience in the search — this contrasts with the role of the improvement phase which is merely to locate a local optimum. Applications of GRASP include vehicle routing[9], and single machine scheduling[10]. A review of other applications, such as flight scheduling for airlines, as well as directions for further reading is given in Feo[8].

Simulated Annealing

Simulated Annealing (SA) is based upon an analogy between optimisation and the annealing process in physics[11]. In terms of neighbourhood search, it can be thought of as a form of stochastic/any-accept hillclimbing with a modified acceptance criterion that accepts lower quality solutions with a probability (the **Metropolis criterion**) given by the equation below:

$$p(accept) = \exp(\frac{quality(s_{curr}) - quality(s_{new})}{T_k})$$

where T_k is the **temperature** at time-step k; this controls how likely it is for a lower quality solution to be accepted, and thus allows SA to escape local optima. The temperature varies with time according to a **cooling schedule** where, usually, the temperature is reduced as the optimisation progresses, to allow **exploration** of the search space at the start of the search, followed by **exploitation** of a promising region later on; the technique-specific choices of initial and final temperatures and the form of the cooling schedule are important so to obtain a balance between exploration and exploitation (also termed **intensification/diversification**). However, there is no reason why the cooling schedule should be of any particular form, or even monotonically decreasing — the choice is problem-dependent. That said, two common cooling schedules are: $T_{k+1} = \alpha T_k$, or $T_{k+1} = T_k/(1 + \beta T_k)$ which generally work well[12]. Work on SA has looked at the theory (eg. Hajek[13]), cooling schedules[12], and applications of SA (eg. sequencing[14,15], timetabling[16], and the Steiner problem in graphs[17]). Further information on SA can be found in a variety of sources[18,19,20].

Threshold Methods

Threshold methods are additional extensions of stochastic/any-accept hillclimbing, that use the idea of a **threshold** which sets a level below which new solutions will not be accepted (i.e. acceptance is deterministic). For similar reasons to SA, the threshold, L_k, is time-varying.

- **Threshold Accepting** (TA) accepts a new solution if its quality is not below a set threshold relative to the current solution[21] (ie. accept if $quality(s_{new}) \geq quality(s_{curr}) - L_k$).

- **Record-To-Record Travel** (RTRT) accepts a new solution if its quality is not below a certain threshold relative to the best solution or **record**, s_{best}, found during the search so far[22] (ie. if $quality(s_{new}) \geq quality(s_{best}) - L_k$).

- The **Great Deluge Algorithm** (GDA) accepts a new solution if its quality is not below an *absolute* quality threshold — the current **water-level**[22] (ie. if $quality(s_{new}) \geq L_k$).

These techniques differ in the way the threshold is used. For all of these techniques it is usual to vary L_k according to the schedule $L_{k+1} = L_k + \alpha$, though there is no reason why others cannot be used. For further details and applications, the reader is referred to the papers[23,24].

Evolutionary Algorithms

Evolutionary Algorithms (EAs) are based upon the theory of evolution by natural selection[25] — a population of solutions is maintained, and allowed to 'evolve. Three main styles of EA exist: **Genetic Algorithms** (GAs)[26], **Evolutionary Programming** (EP)[27], and **Evolution Strategies** (ESs)[28]— but the basic idea behind them is the same and the differences can be considered historical. As an example of the concept, the following describes a simple EA with **steady-state reproduction**:

1. Generate an initial population of solutions.

2. Select two **parent** solutions from the population according to their quality.

3. Apply move operator(s) to generate a new solution, s_{new}.

4. If s_{new} is superior to the worst member of the population, s_{worst}, then s_{new} replaces s_{worst}.

5. Go back to step 2 until the termination criterion is reached.

There are many flavours of EA as almost all of the stages have a wide choice of alternatives; however all have **population-based** acceptance criteria as a *set* of solutions are maintained and a new set of solutions are produced by applying the available move operator(s), and then somehow merging the two sets. Two types of moves are commonly used: **mutation** (roughly analogous to asexual reproduction), which is equivalent to the conventional move operator; and a *binary* move operator, **crossover** (roughly analogous to sexual reproduction) which selects two candidate solutions and (probabilistically) swaps information between them. An example is 'two-point' crossover which randomly picks two points along the strings and swaps the contents of the string between those points, to produce a child as shown in Figure 1.

0 0 | 0 0 0 0 | 0 0 $\xrightarrow{\text{Crossover}}$ 0 0 1 1 1 1 0 0

1 1 | 1 1 1 1 | 1 1 $\qquad\qquad\quad$ 1 1 0 0 0 0 1 1

Figure 1: An Example of Two-Point Crossover

The usual rationale for why crossover is a useful search operator is that the recombinative process can bring together portions of separate strings associated with high fitness to produce even fitter children. Both the population-based nature of the search, and the crossover operator help to avoid the search being trapped in local optima.

Unfortunately, some researchers still equate EAs with encoding solutions as binary strings — due to a misunderstanding of early theoretical work on the **schema theorem**[26]. This is simply not true[29] and successful EA applications use whichever encoding is appropriate to the problem. Such applications are extremely varied, covering fields as diverse as chemistry[30], machine learning[31], and OR. Example applications in OR include: sequencing problems[32], vehicle routing[33], and timetabling[34]. A variety of textbooks are available though Michalewicz[35] is a good starting point for those interested how to apply EAs, whereas the recently-released 'Handbook of Evolutionary Computation' is recommended for its in-depth coverage of the field[36].

Tabu Search

Tabu Search (TS)[37] is based on steepest-ascent or first-ascent hillclimbing, and avoids local optima in a deterministic way based an analogy with memory, which centres on the use of a **tabu list** of moves/solutions (or their features) which have been made/visited in the recent past (**tabu tenure**) of the search. Applications of TS include: vehicle routing[38], graph colouring[39], and path assignment[40]. In TS, the acceptance criterion of the hillclimber is altered slightly so that if no improving move can be found after the neighbourhood has been fully examined, then the move that gives the *least* drop in quality is taken. Then a basic form of memory, **recency**, is used which is *short-term* in nature. In essence, any move/solution that is on the tabu list cannot be made/revisited; this prevents the search **cycling** in an already explored area of the search space.

In addition, **aspiration criteria** can be used to override this mechanism under certain circumstances (eg. when making a tabu move would lead to a higher quality solution). As neighbourhoods can be quite large and thus expensive to search fully, **candidate list** strategies are often used to choose subsets of the neighbourhood to search. An alternative approach is to use some form of cheap, but approximate, evaluation procedure.

A form of *long term* memory, **frequency** can be used to direct the search by adjusting the quality function (eg. solutions closer to a frequently visited area of the search space are penalised more). Other forms of memory are: **quality** which refers to solutions of high quality, and is often used to intensify search in the region of good solutions; and **influence** which is a measure of change in solution structure, and often used as part of an aspiration criterion. Of course, all of these memory structures have to be defined for the problem being solved.

Current research in this area has looked at: dynamic rules for tabu tenure[41], hashing functions to identify tabu solutions[42], and hybrids with other methods[43]. However, a recency-based approach with a simple neighbourhood structure, and a restricted candidate list strategy will often produce good results[4]. For more information, the reader is directed to Glover[44].

Other Meta-Heuristics

Another techniques that fit into the framework of neighbourhood search include **Iterative Repair**[45] which views moves as 'repairs' to flaws in the solutions and has been successfully applied to coordinate space shuttle ground processing, and **Tabu Thresholding**[46] which is a variant of TS that replaces some of the memory structures in TS with some form of randomisation. Needless to say, hybrids of all of the techniques described here are possible. Also, meta-heuristic techniques exist that are not local search-based, though they are less commonly used. However two are worthy of mention. **Ant Systems** are based on the use of pheromone trails by ants[47] — a population of 'ants' makes a path through the solution construction process, where the choice at each stage is balanced between following the trails of other ants, and what a greedy heuristic would decide. Finally, **Hopfield Nets** arose from neural network research on associative memories and can also be used for optimisation[48].

Which to Use? The Methodological Gap

In practice, this choice of techniques results in the user facing the difficult question of how to design an effective optimiser. Theoretical work, the **No Free Lunch (NFL) theorem**[49], argues that no optimiser can outperform another over the entire space of problems, thus viewing black-box optimisers as the optimisation communities' philosopher's stone — for an optimisation technique to be effective it *must* contain domain knowledge, even implicitly. However, it is not enough just to say that domain knowledge is important, guidelines are required on *how* to approach a problem, extract the salient features, and to map these onto the optimiser. Therefore the design of local search optimisers remains very much an art, and a 'methodological gap' remains between the literature, textbooks, and the application of these techniques which needs to be addressed. The remainder of this paper will propose a view of neighbourhood search, based on the idea of a **Knowledge Based System** (KBS)[50], and use it to review ways that domain knowledge can be incorporated into optimisation.

A Change in Viewpoint: A KBS Approach

A KBS is simply[50] "a computer system that represents and uses knowledge to carry out a task". This captures our aims, but it is not in itself useful as any working system must, from the NFL theorem, possess domain knowledge in some form. Fortunately, the AI community has, for many years, been addressing the problems of how to design a KBS, and therefore if we could place neighbourhood search optimisers into a KBS framework, then the reuse of the extensive research into such systems may be possible.

One such useful concept from the KBS literature is the distinction made[51] between the **knowledge level**, that is the description of *what* domain knowledge is present in the system, and the **symbol level** which is the description of *how* the knowledge is represented in the system (ie. its data structures and the operations on them). This allows us to group the various extensions used in the optimisers described here as different symbol level implementations of the same piece of domain knowledge, thus allowing a more unified and declarative view of these techniques quite separate from the common implementational framework shown in the introduction.

That said, every KBS technique exploits different types of knowledge, often in different ways, and thus if a knowledge engineer was to use neighbourhood search, he would need to know what questions to ask the domain expert. Therefore the *roles* that domain knowledge plays in neighbourhood search need to be defined. To this end, this paper proposes that domain knowledge can be split into three roles:

- **Problem Solving Knowledge** — this knowledge assists the search by providing *problem-specific* knowledge and problem-solving to guide the search.
- **Problem Specification (Goal) Knowledge** — specifying what desirable solution(s) are (ie. the evaluation function).
- **Search Control Knowledge** — given a search space, how do we go about searching it? Our knowledge of the search process is represented here.

Once these roles have been defined, then there are various types of **knowledge sources** available that need to be identified — the overall structure is summarised in Figure 2. These knowledge sources can be expressed in concrete terms that a non-specialist in optimisation can understand, and represented to the optimisation algorithm in such a manner that a plan of incremental improvement and experimentation is possible. This is because, in practice, the user examines a problem and expresses his 'beliefs' about the problem domain, which are then incorporated into the optimiser to experimentally test whether the domain knowledge is correct. In addition there is scope to include the user as an *interactive* knowledge source[52]. The remainder of this paper will briefly justify, identify, and review the knowledge sources available.

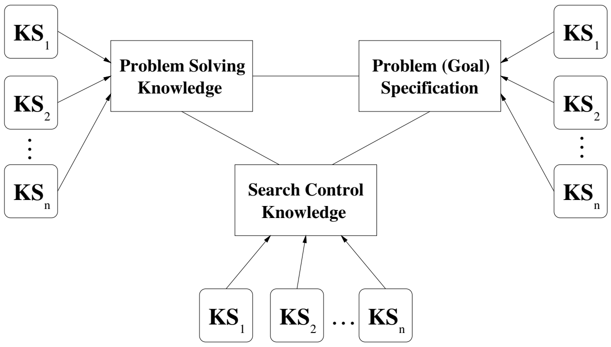

Figure 2: Knowledge Roles in Neighbourhood Search

Where to Start? Landscape vs Search Algorithm

In order to define our knowledge sources it is necessary to first justify the categorisation of knowledge sources and then determine the relative importance of them in respect to the operational criterion of how easily they can be used to quickly design an effective optimiser.

Recent work has examined this issue from first principles[53]. To this end, neighbourhood search optimisers were split into two components: the first is the *landscape*, described earlier, which is the combination of the solution encoding, neighbourhood operators, and objective function (at the knowledge level these would correspond to problem-solving and goal knowledge); the second is the *optimisation algorithm* which searches the landscape (at the knowledge level this would correspond to an implementation of the search control knowledge).

This formalism allows domain knowledge to be placed into the optimiser in two ways: the first is to transform the landscape so that a given optimisation algorithm can find high quality solutions sooner; the second is to arbitrarily fix the landscape and to devise an algorithm that visits the high quality solutions sooner. However, the entire space of algorithms is not being considered, which constrains the changes that can be made to the algorithm to improve search performance, as noted earlier, as all neighbourhood search algorithms rely upon the landscape being correlated.

In addition, it is easier in practice to derive a good fitness landscape from the problem, than a good search algorithm. This is the case for two reasons: the first is that it is widely acknowledged [2] that the problem of matching an optimiser to a landscape is extremely hard — in fact, no effective methods of characterising search spaces so to predict algorithm performance is available; the second is that it has often been observed[35] that 'natural' encodings and operators for a problem, that are relatively straightforward to derive, give good results in applications.

The natural correspondences highlighted above, suggest that the classification used here is justified, and provides a reason for focusing our attention, at least early on, to the acquisition and representation of problem-solving and goal knowledge. Finally, the rationale behind the design of an effective landscape is based upon assumptions that are common to all local search optimisers, so results obtained with one (eg. hillclimbing) should be readily transferable across optimisation techniques[53].

Problem-Solving Knowledge

What exactly *is* meant by problem-solving knowledge? A working definition is that it is all of the knowledge that can be directly related to the problem itself, that is not involved with specifying the quality of a solution. This allows for a clean separation from the technique-specific aspects of the search algorithm, and is in fact wider in scope than just the fitness landscape — it is possible to identify the following types of knowledge sources:

1. Problem features that correlate with solution quality.
2. Decomposition of the problem into more tractable subproblems.
3. Areas of the search space which can be excluded from the search.
4. Areas of the search space where good solutions lie.
5. Prediction of improving moves.

It is important to note that of these roles, the first of these is the most important to get right; as all of the others require a correlated landscape in order to work effectively. In addition, these knowledge sources can be also transferred, by similar arguments to above, across optimisers[53].

The Fitness Landscape. This relates to the design of the fitness landscape, which has been already discussed, and arises from the problem features that the problem encoding and operators manipulate (eg. edges in the 2-opt operator for the TSP). These features can be modelled, at the knowledge-level, by equivalence relations with theory that provides formal specifications for suitable solution encodings and neighbourhood operators at the symbol level[54].

Simplifying the Problem. The next two knowledges sources concern themselves with how to simplify the problem itself. The first expresses the user's knowledge about how to split the problem up into subproblems, an example of this would be 'divide and conquer' EAs that encode each subproblem separately, and the method by which they are brought together for evaluation[55]. The second focuses upon solutions that can be excluded from the search. An example arises in constraint satisfaction problems, where it is possible with the use of consistency algorithms[56], to use the constraints to reduce the search space.

Intelligent Initialisation. It is common to start the search with a heuristically generated solution; in the context of the framework discussed here, this effectively is a representation of the user's belief that this solution lies close to other, higher quality, solutions. This requires a correlated landscape to be effective, as the belief that a good initial solution is in the vicinity of better solutions is only realisable if the landscape is correlated (as correlation implies that good solutions are closer to each other than poor solutions).

Directing The Neighbourhood. For each solution in the search space, there are number of 'moves' to other solutions available, most of which are non-improving. Conventional implementations examine these moves in a random order; However if a method existed that could cheaply determine which moves were improving, then this would result in high-quality solutions being found more quickly as non-improving moves then need not be evaluated. Fortunately, it is often possible to devise heuristics about which moves would be most likely to be improving. Such a heuristic has been successful for timetabling problems[57], where exams with a high number of constraint violations were more likely to be moved. This knowledge source can be mapped to the optimiser in two ways: the first is to use a TS candidate list strategy to select a biased subset of the possible moves to try; the second approach is the 'directed mutation' approach in the example above. Either way, a correlated landscape is needed for this to be effective as these methods allow the optimiser to merely hillclimb faster by removing unnecessary moves — if the search space was uncorrelated, then local optima would just be found more quickly.

Goal and Search Control Knowledge

Problem Specification (Goal) Knowledge defines what the desired solution is by stating the relative quality of the solutions. In neighbourhood search, the objective function fulfils this role. This is important to get right, otherwise the wrong problem will be optimised! The traditional approach to optimisation, is to build a mathematical model of the problem and then to solve it, suffers from the disadvantage that such a model is an approximation of the actual problem domain[58]; though local search, in common with some other heuristic methods, does allow for a rich representation of the user's preferences[4]. Fortunately, the literature does provide some assistance in this task, such as utility theory[59], and it usually is possible in practice to produce an acceptable model — although this may not necessarily be straightforward, a good example is in the area of multi-objective optimisation[60] where it is often difficult in practice for the end-user to specify the relative importance of the objectives.

Search Control Knowledge. By default, all of the technique-specific additions that control and parameterise the various techniques reviewed earier fall under this class. In practice, as noted earlier, knowledge acquisition is largely a process of experimentation (tuning), as the mapping between optimisation technique parameters and the landscape being search is usually *implicit* and poorly characterised. That said there are commonalities between the techniques. An example of this would be if, at the knowledge level, the landscape was thought to be quite rugged but otherwise correlated, then some mechanism to hop over the local peaks would be a good idea. So if simulated annealing was used, then the temperature term would perform this function, with the required temperature increasing as the ruggedness increases; with tabu search, this knowledge would be represented in the tabu list, aspiration criteria, etc. so as to strike the correct balance between exploration and exploitation.

Conclusion

This paper has concerned itself with providing an introduction to the neighbourhood search paradigm and the main techniques in use today in OR. An overview of these techniques was then given, followed by an overview of ways of incorporating domain knowledge into them, based around a KBS framework. The above framework is important in the context of local search because, despite the realisation that domain knowledge is necessary, very little work as yet has attempted to systematise the roles that knowledge plays in these techniques. Notable exceptions include[35,61], though discussion in both is grounded in the 'symbol level', rather than in the 'knowledge level' which is the usual level of description advocated for KBSs, and the coverage of knowledge sources in these works is somewhat restricted.

Finally, it is the author's view that further research into the identification and formalisation of knowledge sources, and the formulation of effective design methodologies in a common framework is essential if the neighbourhood search community is to make further progress in transferring these techniques to real-world applications.

Acknowledgements

I would like to express my thanks to the Engineering and Physical Sciences Research Council (EPSRC) for their support via a research studentship (95306458), and to my PhD supervisors, Peter Ross and Tim Duncan, for their advice and encouragement.

References For Further Reading

1. Michael R. Garey and David S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. Freeman, 1979.

2. T. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, University of New Mexico, 1995.

3. V. J. Rayward-Smith. A Unified Approach To Tabu Search, Simulated Annealing and Genetic Algorithms. In *The Proceedings of the UNICOM Seminar on Adaptive Computing and Information Processing*, 1994.

4. C. R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publications, 1993.

5. I. H. Osman. An introduction to Meta-Heuristics. In M. Lawrence and C. Wilsdon, editors, *Operational Research Tutorial Papers*, pages 92–122. Operational Research Society, 1995.

6. I. H. Osman and J. P. Kelly. *Meta-Heuristics. Theory and Applications*. Kluwer Academic Publishers, 1996.

7. I. H. Osman and G. Laporte. Metaheuristics for combinatorial optimisation problems: An annotated bibliography. *Annals of Operational Research*, 63:513–628, 1995.

8. T. A. Feo, J. F. Bard, and R. F. Claflin. An overview of GRASP methodology and applications. Technical report, University of Texas at Austin, 1991.

9. C. A. Hjorring. *The Vehicle Routing Problem and Local Search Metaheuristics*. PhD thesis, The University of Auckland, New Zealand, 1995.

10. T. A. Feo, K. Venkatraman, and J. F. Bard. A GRASP for a Difficult Single Machine Scheduling Problem. *Computers Ops. Res.*, 17(8):635–643, 1991.

11. S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

12. M. Lundy and A. Mees. Convergence of an annealing algorithm. *Math. Prog.*, 34:111–124, 1986.

13. B. Hajek. Cooling schedules for optimal annealing. *MOR*, 13:311–329, 1988.

14. I. H. Osman and C. N. Potts. Simulated annealing for permutation flow-shop scheduling. *OMEGA*, 17:551–557, 1989.

15. F. A. Ogbu and D. K. Smith. The application of the simulated annealing algorithm to the solution of the $n/m/P/C_{max}$ flowshop problem. *Computers and Ops. Res.*, 1990.

16. D. Abramson. Constructing school timetables using simulated annealing: sequential and parallel algorithms. *Management Science*, 37:98–113, 1991.

17. K. A. Downsland. Hill-climbing, simulated annealing, and the Stenier problem in graphs. *Eng. Opt.*, 17:91–107, 1991.

18. N. E. Collins, R. W. Eglese, and B. L. Golden. Simulated annealing — an annoted biblography. *AJMMS*, 8:209–307, 1988.

19. P.J. M. VanLaarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. Kluwer, Dordrecht, 1988.

20. E. H. L. Aarts and J. H. M. Korst. *Simulated Annealing and Boltzmann Machines*. Wiley, Chichester, 1989.

21. G. Dueck and T. Scheuer. Threshold Accepting: A General Purpose Optimisation Algorithm Superior to Simulated Annealing. *Journal of Computation Physics*, 90:161–175, 1990.

22. G. Dueck. New Optimisation Heuristics: The Great Deluge Algorithm and the Record-to-Record Travel. Technical report, IBM Germany, Heidelburg Scientific Center, 1990.

23. I. Althofer and K. U. Koschnick. On the convergence of 'threshold accepting'. *Applied Mathematics and Optimisation*, 24:183–195, 1991.

24. M. Sinclair. Comparision of the performance of modern heuristics for combinatorial problems on real data. *Computers and Operations Research*, 20:687–695, 1993.

25. C. Darwin. *On the Origin of Species*. John Murray, London, 1859.

26. John H. Holland. *Adaptation in Natural and Artificial Systems*. Ann Arbor: The University of Michigan Press, 1975.

27. L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence Through Simulated Evolution*. John Wiley and Sons, Inc., 1966.

28. I. Rechenburg. *Evolutionstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. Frommann-Holzberg, 1973.

29. N. J. Radcliffe. Non-Linear Genetic Representations. Technical report, Edinburgh Parallel Computing Centre, 1992.

30. Hugh M. Cartwright and Stephen P. Harris. Analysis of the distribution of airborne pollution using genetic algorithms. *Atmospheric Environment*, 27:1783–1791, 1993.

31. David E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Reading: Addison Wesley, 1989.

32. C. R. Reeves. A genetic algorithm for flowshop sequencing. *Computers & Ops. Res.*, 22:5–13, 1995.

33. S. R. Thangiah. An adaptive clustering method using a geometric shape for vehicle routing problems with time windows. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 536–544, San Francisco, Ca., 1995. Morgan Kaufmann.

34. D. Corne, H.-L. Fang, and C. Mellish. Solving the Module Exam Scheduling Problem with Genetic Algorithms. In Paul W. H. Chung, Gillian Lovegrove, and Moonis Ali, editors, *Proceedings of the Sixth International Conference in Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 370–373. Gordon and Breach Science Publishers, 1993.

35. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Artificial Intelligence. Springer-Verlag, New York, 1992.

36. T. Baeck, D. B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

37. F. Glover. Tabu Search: A Tutorial. *Interfaces*, 4:445–460, 1990.

38. F. Semet and E. Taillard. Solving real-life vehicle routing problems effectively using taboo search. *Annals of Ops. Res.*, 41, 1993.

39. A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 29:345–351, 1987.

40. S. Oliveira and G. Stroud. A parallel version of tabu search and the path assignment problem. *Heuristics for Combinatorial Optimisation*, 4:1–24, 1989.

41. R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 1994.

42. M. Hasan and I. H. Osman. Local search algorithms for the maximal planar layout problem. *International Transactions in Operations Research*, 2:89–106, 1995.

43. I. H Osman and N. Christofides. Capacitated clustering problems by hybrid simulated annealing and tabu search. *International Transactions in Operations Research*, 1:317–336, 1994.

44. F. W. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

45. M. Zweben, E. Davis, B. Daun, and M. Deale. Scheduling and Rescheduling with Iterative Repair. *IEEE Transactions on Systems, Man, and Cybernetics*, 1993.

46. F. Glover. Tabu Thresholding: Improved Search by Nonmonotonic Trajectories. *ORSA Journal on Computing*, 7(4):426–442, 1995.

47. M. Dorigo and L.M. Gambardella. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

48. J. Hertz, K. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, 1991.

49. D.H. Wolpert and W.G. Macready. No free lunch theorems for search. Technical report, SFI-TR-95-02-010, Santa Fe Institute, 1995.

50. M. Stefik. *Introduction to Knowledge Systems*. Morgan Kauffmann, 1995.

51. A. Newell. The Knowledge Level. *Artificial Intelligence*, 18(1):87–127, 1982.

52. A. L. Tuson, P. Ross, and T. Duncan. On Interactive Neighbourhood Search Schedulers. In *the 16th UK Planning and Scheduling SIG*, 1997.

53. A. L. Tuson, P. Ross, and T. Duncan. Paying for lunch in neighbourhood search. To be presented at *Combinatorial Optimization '98*, 1998.

54. N. J. Radcliffe. Equivalence class analysis of genetic algorithms. *Complex Systems*, 5(2):183–205, 1991.

55. L. Gonzales-Hernandez. Evolutionary Divide and Conquer for the Set-Covering Problem. Master's thesis, Department of Artificial Intelligence, University of Edinburgh, 1995.

56. E. P. K. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London and San Diego, 1993.

57. P. Ross, D. Corne, and H.-L. Fang. Improving evolutionary timetabling with delta evaluation and directed mutation. In Y. Davidor, H-P. Schwefel, and R. Manner, editors, *Parallel Problem-solving from Nature — PPSN III*, LNCS, pages 566–565. Springer-Verlag, 1994.

58. H. M. Wagner, M . H. Rothkopf, C. J. Thomas, and H. J. Miser. The Next Decade in Operations Research: Comments on the CONDOR report. *Operations Research*, 37(4):664–672, 1989.

59. R. Keeney and H. Raiffa. *Decisions with Multiple Objectives, Preferences, and Value Tradeoffs*. John Wiley, New York, 1976.

60. C. M. Fonseca and P. J. Fleming. An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation*, 3(1):1–16, 1995.

61. M. Nussbaum, M. Sepulveda, M. Singer, and E Laval. An architecture for solving sequencing and resource allocation problems using approximation methods. *Journal of the Operational Research Society*, 49(1):52–65, 1998.