MODERN HEURISTIC TECHNIQUES

V Rayward-Smith

Rapporteur: Dr Rogério de Lemos

MODERN HEURISTIC TECHNIQUES

V.J. Rayward-Smith
School of Information Systems
University of East Anglia
Norwich
NR4 7TJ

Abstract

The paper reviews the most important of the so-called, modern heuristic techniques – tabu search, simulated annealing and genetic algorithms. They are all presented as simple extensions to the local search paradigm. One advantage of viewing them in this way is that further variations become apparent together with techniques for hybridisation. These are briefly explored.

The paper also includes two case studies: – the classic, Steiner tree problem and a very practical use in Data Mining.

1 Introduction

Let us assume that we have a universe, U, of (potential) solutions to a problem. We seek solutions that meet certain constraints, or, put another way, that lie in some subset, S, of U. Assuming each problem can be assigned a value in some totally ordered set, W, an optimisation problem will seek one or more elements of S of maximum value. Thus, given a set $S \subset U$, a totally ordered set, W (often the reals, $\mathcal{R}$) and an objective function, $value : U \rightarrow W$, an optimisation problem is of the form

$$\begin{array}{c} \text{maximise } value(u) \\ \text{such that } u \in S \subset U. \end{array}$$

A common technique used in optimisation is to "price out" the constraints. This is done by defining $value$ on all elements of U in such a way that any $u \in U \setminus S$ has such a small value that it is guaranteed not to provide the maximum value. In the case where $W = \mathcal{R}$, this is usually achieved by ensuring $value(u)$ is a large negative number for any $u \in U \setminus S$. This technique can also be used to cope with any "soft" constraints, i.e. constraints that are desirable but not absolutely essential. The optimisation can then be expressed simply as

$$\begin{array}{c} \text{maximise } value(u) \\ \text{such that } u \in U. \end{array}$$

As an example, consider the well known, NP-hard, *0-1 Knapsack Problem*, viz.

0-1 KNAPSACK (KP)
Given: n items, $1, 2, \ldots, n$, where item i has associated profit, $p_i \in Z^+$ and weight, $w_i \in Z^+$, and a knapsack of capacity $C \in Z^+$.
Problem: Find a subset, $A \subset \{1, 2, \ldots, n\}$ to
maximize

$$Profit[A] = \sum\{p_i \mid i \in A\}$$

subject to

$$Weight[A] = \sum\{w_i \mid i \in A\} \leq C.$$

This problem may be replaced by

Problem: Find a subset, $A \subset \{1, 2, \ldots, n\}$ to
maximize

$$Profit[A] = \sum\{p_i \mid i \in A\} - \lambda \max\{\sum\{w_i \mid i \in A\} - C, 0\},$$

where $\lambda \in Z^+$ is a large positive number.

A solution to this problem can be represented by a binary string u of length n, where $u_i = 1$ if $i \in A$ and $u_i = 0$ if $i \notin A$. Any solution, u, then has

$$value(u) = \sum_{i=1}^{n} p_i u_i - \lambda \max\{\sum_{i=1}^{n} w_i u_i - C, 0\}.$$

2 Neighbourhood Search

A neighbourhood search exploits one (or more) neighbourhood functions of the form

$$neighbour : U \rightarrow 2^U$$

which, given a solution, $u \in U$, delivers a set of solutions "close to" u. A pool of solutions is maintained and, at each iteration of the algorithm, one or more elements of the pool are expanded by applying the *neighbour* function.

In the simplest forms of neighbourhood search, the pool size is maintained at one. The algorithm proceeds by continually replacing u by a neighbour of greater value than u. In a simple *greedy hill climbing* algorithm u will always be replaced by the first neighbour v found in $neighbour(u)$ such that $value(v) > value(u)$. In a *steepest ascent* algorithm, the whole of $neighbour(u)$ is explored to find a $v \in neighbour(u)$ such that

$$\begin{aligned} value(v) &> value(u) \text{ and} \\ value(v) &\geq value(w)\ \forall w \in neighbour(u). \end{aligned}$$

Between these two extremes, the algorithm designer can opt for a partial exploration of $neighbour(u)$ to find the next solution.

It is also common to maintain a pool of solutions of size greater that 1. One or more neighbours of any solution in the pool can be added to the pool replacing the solutions whose neighbourhoods have been explored or which the algorithm deems will no longer be worth maintaining.

Crucial to the success of a neighbourhood search algorithm are the following.

Representation The algorithm designer must determine how each $u \in U$ is represented. This representation should ensure that the neighbourhood function(s) and the evaluation function can be efficiently implemented.

Initialisation The initial pool of one or more solutions must be determined. This may be achieved using a random generating function or by the use of one or more heuristics.

In the GRASP approach, the generation of an initial pool of reasonable solutions is critical to the success of the algorithm.

Neighbourhood Function(s) The function $neighbour : U \rightarrow 2^U$ must be specified. There may be several choices and sometimes more than one is used at each step. Alternatively, a choice is made at each step and which one is used will change from one step to the other. In an *adaptive* neighbourhood search, the algorithm monitors the success rate of the various neighbouhood functions and selects (usually probabilistically) the most promising neighbourhood function. In a *variable depth search*, the algorithm uses $neighbour$, $neighbour^2$, ..., $neighbour^k$ as neighbour functions. Typically $k < 4$ and the higher order neighbours are only brought into play when the lower order neighbours are failing to find improved solutions.

Different neighbourhood functions can be applied to various solutions. This may have implications concerning the representations used. It may be the case that

one neighbourhood function is far more efficient if representation A is used while another requires representation B. In such cases, it is sometimes worthwhile incorporating functions to switch representations and to exploit these within the algorithm (see figure 1). In this diagram, we have just two possible representations, rep A and rep B. ϕ_{AB} changes the representation of a solution from rep A to rep B and ϕ_{BA} vice versa. We assume $neighbour_1$ works most efficiently on rep A and $neighbour_2$ on rep B. Which representation to use will then be determined by the probability of using $neighbour_1$ rather than $neighbour_2$ and the efficiency of computing the evaluation function, $value$, on the two representations.

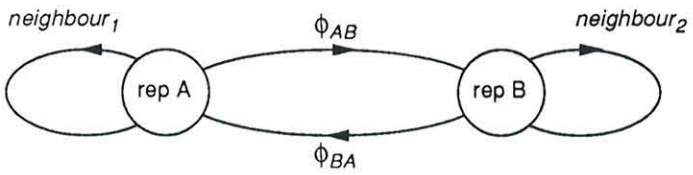

Figure 1: Changing representations.

Pool Size The size of the maintained pool is often one but, especially in distributed and parallel systems, a larger pool size is common. If the pool size is greater than one, a strategy is required to *select* those solutions from the pool which will be expanded.

Selection Criteria If the size of the pool is greater than 1, a decision has to be made as to which solutions are to be expanded. This could be all of them, a random selection or a subset chosen by some heuristic or expert system.

Acceptability Criteria Once it is determined that a node should be expanded, the algorithm needs to have criteria for determining when one or more neighbours are acceptable. Simple criteria are

First Found The first neighbour, $v \in neighbour(u)$, such that $value(v) > value(u)$ is acceptable.

All Better All neighbours $v \in neighbour(u)$ such that $value(v) > value(u)$ are acceptable

All Best All neighbours $v \in neighbour(u)$ such that $value(v) > value(u)$ and $value(v) \geq value(w)$ $\forall w \in neighbour(u)$ are acceptable

Best Any single solution from the set of all best solutions is acceptable

In the above, we assume we always require an improvement in the value. This can be relaxed so that the criteria is $value(v) \geq value(u)$ or even $|value(v) - value(u)| < \epsilon$ for some small ϵ.

If the acceptability criteria are totally relaxed, so that any solution is acceptable we have a *random search*. In the case where the pool size is 1 and we simply move from one solution to an arbitrary neighbour, this is known as a *random walk*.

Replacement Criteria With a pool size $n > 1$, the selection criteria selects $m \leq n$ solutions for expansion. Having determined the neighbourhood functions and the acceptability criteria to be used, we then have k new solutions to incorporate into the pool. There are many possible options including

Keep All All acceptable neighbours are added to the pool and expanded solutions are removed.

m-Best The m expanded solutions are removed and, assuming $k > m$, the m new solutions of highest value are used.

Diversify If $k > m$, some heuristic is applied to select m solutions representing a diverse collection of solution types to replace the expanded nodes.

Intensify If $k > m$, a subset of m solutions which have common characteristics are used to replace the expanded nodes.

In practice, diversification is used at the beginning of the search and intensification only at the latter stages.

Halting The algorithm must incorporate some halting criterion. This might be

Time-up Some time limit is imposed for the search.

Iteration Limit A limit on the number of iterations is imposed.

No new solutions There are no acceptable solutions found in an iteration.

No significant improvement The rate of improvement of solutions is very slow.

Neighbourhood search is often a very successful technique to solve difficult, large-scale, commercial problems. A common criticism of the method is that, even if the algorithm is run until there are no more acceptable solutions, the best solution found will not be close to the optimal. The algorithm tends to get stuck in a *local optimum* – see figure 2. Although this criticism is fair, it is also widely misunderstood and all

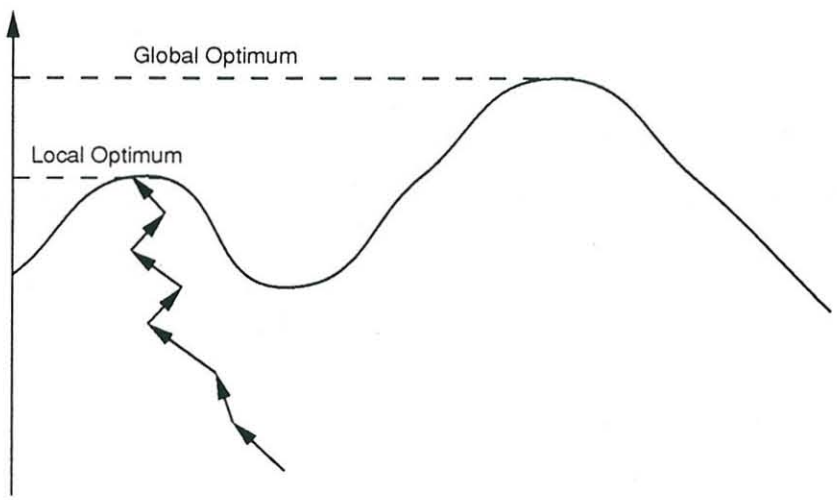

Figure 2: Finding a local optimum

too often the approach is dismissed without adequate exploration.

The first point to make is that the representation and the neighbourhood function being used are critical to the topology of the search space. As an illustration of this, consider the task of maximizing the function $f : X \to R$ where $X = \{0, 1, 2, 3, 4, 5, 6, 7\}$ and

$$f(x) = -(x-4)^2.$$

As a function on the reals $x \mapsto -(x-4)^2$ is a smooth function with a maximum value of $x = 4$. However, if we represent each $x \in X$ by a 3-bit binary string, set $value = f$ in the obvious way, and assume the neighbourhood function is a simple bit change, we find $x = 3$ is a local optima. (f(011) = -1 and f(111) = - 9, f(001) = - 9 and f(010) = - 4.)

Now, let us use the same representation but use a neighbourhood function obtained by simply adding/subtracting 1 mod 7 from the binary number. In this case, there are no local optima.

Note, there is also scope to choose a neighbourhood function which partitions the search space into distinct regions. In our example, a neighbourhood function which flips two bits will partition the search space into

$$\{000, 110, 011, 101\} \text{ and } \{001, 111, 010, 010\}.$$

In general, if we define

$$neighbour^* : U \to 2^U$$

recursively by

$$neighbour^*(u) = neighbour(u) \cup neighbour^*(neighbour(u))$$

then $neighbour^*$ partitions the search space into equivalence classes. If a global optimum is to be found, we must search all of these equivalance classes.

When it appears that the local search procedure has got stuck in a local optimum, there are various steps that can be taken if the search is not to be terminated. The major techniques are

Change Topology Select a different neighbourhood function (and possibly representation).

Start Again Find an unexplored solution and begin again from that.

Relax Acceptability Criteria Allow some 'down hill' moves for a limited time in an attempt to get out of the local optimum.

There are three common variants of local search which have been particularly good at escaping from poor quality local optima and are widely used to solve real-world optimization problems. Simulated Annealing is a technique for a controlled relaxation of the acceptability criteria. Tabu Search is an adaptive, constrained random walk. Genetic Algorithms always have a pool of more than one solution and extend the neighbourhood function to a binary function

$$child : U \times U \to U.$$

We will discuss these in detail in the next section.

{objective is to maximise $value(p)$ such that $p \in U$ }
u : solution $\in U$;
$Tabu$: set of rules, each rule of type $U \rightarrow \{true, false\}$;
$initialise(u)$:
$initialise(Tabu)$; {very often to Φ}
while not $finish(u)$ **do**
    begin
    $u := select(N)$
    where $N = \{n \mid n$ is a neighbour of u and $f(u)$ =false $\forall u \in Tabu\}$;
    $updateTabu$
    end
end {Tabu Search}

Figure 3: The Tabu Search Paradigm

3 Tabu Search

In tabu search, the search moves from one solution to a neighbouring solution but omits those specified to be "tabu". The "tabu list" is a dynamic set of user-defined rules which defines those neighbours which are tabu. A good introduction to this paradigm can be found in [32, 33, 36]. Representing this tabu list by $Tabu$, tabu search method can be summarised as in figure 3.

In its simplest application, *select* may simply deliver a random element from N. The move is then not necessarily to a better solution. Alternatively, in an attempt to move to a better solution, a small "promising" part of the neighbourhood can be explored; if this does not yield an reasonable valued solution then a next, most promising subneighbourhood can be explored, etc.

In practice, we may work with more than one element and maintain a pool of solutions, deciding at each iteration, which one(s) to expand. Although there is considerable scope for constructing variants of the basic tabu search paradigm, the essential feature is the exploitation and updating of the tabu list.

The sort of rules we might find in the tabu list include the following.

1. v is tabu if v was the predecessor of u.
2. v is tabu if v has been visited recently.
3. Assuming a binary representation, v is tabu if v is obtained by changing a bit changed in the previous iteration.
4. Assuming a binary representation, v is tabu if v is obtained by changing a bit changed recently.
5. v is tabu if $value(v) = value(u)$.
6. v is tabu if $value(v) = value(w)$ for some w which has recently been visited.

Within tabu search, there is then plenty of scope to experiment with the options for defining "recently" and this has been an active area of research. There is also

enormous scope for developing various tabu rules both of a general nature and of a very problem dependent nature. Two types of general rule have been identified; *diversification* attempts to broaden the search space whilst *intensification* narrows the search space.

Tabu rules specify which moves cannot be made. It may also be useful to have rules to specify which solutions are desirable. *Aspiration* rules are used to override the tabu rules but this idea can be generalised. We can envisage an expert system, defined by a set of rules, guiding the search. Each rule has an associated weight, negative if tabu, positive if an aspiration. The combined set of rules thus associates a weight to each neighbour. A large positive weight suggests it is a desirable move, a large negative weight suggests it can be discounted. The difficulty is then constructing this set of rules – considerable expertise is required. However, if this is available, the resulting search can be very efficient. Even if only limited expertise is available, it would seem highly desirable to include it in any search algorithm. Thus although we might not be able to fully rely on a rule based approach, it is desirable to incorporate some such expertise in a search algorithm.

Tabu search is in its infancy; most implementations concentrate on the tabu aspects and have yet to exploit the more general rule based approach. Nevertheless, some excellent results have been obtained. These include the following application areas.

- Packing and Scheduling Problems [2, 17, 20, 30, 34, 55, 81, 91, 92].
- Travelling Salesman and Vehicle Routing [58, 68, 79].
- Telecommunications [3, 66].
- Graph Problems [16, 31, 35, 43].
- Quadratic Assignment [11, 22, 80, 82].

A rule based approach to searching can be very effective and researchers in the field often claim that tabu search outperforms the other techniques we will be discussing. The difficulty with the technique is that considerable expertise and experimentation is required to construct the rules and to ensure its dynamic nature is correctly controlled. Surely the best way forward is to incorporate some rule based search into other search techniques combining the best of all the techniques.

4 Simulated Annealing

Simulated annealing, as a mechanism for searching the feasible search space of an optimisation problem, was first proposed by Kirkpatrick [52] and, independently, by Cerny [10]. Their algorithms were based on that of Metropolis et al. [60] which simulated the controlled cooling of a material by a process now known as *annealing*. The simulated annealing technique is essentially local search in which a move to an inferior solution is allowed with a probability which decreases, as the process progresses, according to some Boltzmann-type distribution. Research involving simulated annealing invariably concerns itself with some aspect of the proof of convergence of the algorithm, or with the application of the technique to a particular problem, and the subsequent search for the optimum set of parameters to use within the algorithm. Recently, a large number

of papers have appeared dealing with the successful application of SA to a wide variety of optimisation problems.

The inspiration for the SA approach to optimisation is the *law of thermodynamics* which states that at temperature, t, the probability of an increase in energy of magnitude, δE, is given by

$$P[\delta E] = \exp(-\delta E/kt), \tag{1}$$

where k is the physical constant known as *Boltzmann's constant*. This equation can be used in a simulation of a system that is cooling until it converges to a steady, "frozen" state. Having generated a perturbation from the current state, the resulting energy change is calculated. If the energy has decreased, the system moves to this new state; otherwise, the new state is only accepted with probability given by equation 1. This cycle can be repeated for a fixed number of iterations at each temperature. Then the temperature can be reduced and the same number of cycles repeated for the new lower temperature. This whole process is then repeated until the system freezes into its steady state.

Now, let us associate solutions of an optimisation problem with the system states. The cost of a solution corresponds to the concept of energy and moving to any neighbour corresponds to a change of state. This is how both Kirkpatrick [52] and Cerny [10] developed the SA approach for optimisation problems.

A simple sequential version of the SA paradigm for optimisation problems can be described as in figure 4. A detailed overview of SA can be found in [24].

```
u := u0 where u0 is some initial solution;
temp := temp0 where temp0 is some initial temperature;
while not finish(u) do
    for i := 1 to n do
        begin
        randomly select u', a neighbour of u;
        improvement := value(u') − value(u);
        if improvement > 0 then u := u'
        else
            begin
            generate x ∈U[0,1];
            if x < exp(improvement/temp) then u := u'
            end {else}
        end {for};
    t := reduce(t)
end {while}
```

Figure 4: A Simple Version of the Simulated Annealing Paradigm

Considerable effort is expended in simulated annealing algorithms to get the cooling rate correct so that the degree of randomness introduced into hill climbing is beneficial.

Nearly all the research reported on SA has used the sequential version of the algorithm although there is some research into parallel SA, e.g. [93]. Successful applications of SA have included the following.

- graph algorithms [12, 23, 46, 47].
- packing and scheduling problems [1, 8, 9, 54, 64, 65, 69, 85, 87].
- travelling salesman and vehicle routing [6, 68, 78].
- quadratic assignment problems [15].

Essentially, simulated annealing has taught us that the inclusion of "downhill" moves in a neighbourhood search algorithm is a good idea provided that the probability of making such moves is suitably reduced during the execution of the algorithm. There is obvious scope for a hybrid algorithm which uses the tabu list to limit possible moves together with simulated annealing techniques. A similar concept is used in thermostatistical persistency [13].

In the simple version of SA described above, the cooling schedule was *monotonic*, that is it never increased. Recent research has considered non-monotonic cooling strategies or reannealing. One common method is to increase the temperature to half its original value whenever it appears the algorithm has got stuck in a local optimum.

5 Genetic Algorithms

The genetic algorithm (GA) paradigm was first conceived by John Holland (University of Michigan) in the late 1960s. It was here that the technique was developed and the foundations of the theory laid for others to build on, see [37, 44]. Genetic algorithms are search techniques, based on an abstracted model of Darwinian evolution. Solutions are represented by fixed length strings over some alphabet (the "gene" alphabet) and each such string is thought of as a "chromosome". The value of the solution then represents the "fitness" of the chromosome. We then apply the "survival of the fittest" principle to a pool of chromosomes and allow the better solutions to combine to produce (hopefully fit) offspring.

The genetic algorithm (GA) paradigm is given in figure 5. The algorithm initialises some pool, or population, of solutions. Then, at each iteration, a subpopulation is selected for "breeding". From this subpopulation, children are generated according to genetic operators. The children are then merged into the original population and the process is repeated. The various genetic algorithms will vary in the precise definition of the five functions, *initialise*, *finish*, *select*, *create* and *merge* but to be a genetic algorithm, it is essential that

- populations have cardinality > 2,
- *create* uses genetic operators such as crossover and mutation (see below), and
- *select* and *merge* depend on the evaluation of solutions and together favour the preservation of solutions with higher *value*.

```
{objective is to maximise value(u) such that u ∈ U }
P, Q, R : multiset of solutions ⊂ U;
initialise(P);
while not finish(P) do
    begin
    Q := select(P);
    R := create(Q);
    P := merge(P, Q, R)
    end
end
```

Figure 5: The Genetic Algorithm Paradigm

The initial population is often generated randomly but sometimes it is "seeded" with known good solutions, perhaps generated by some heuristic. The GA will generally finish after some given number of iterations or after some given time has elapsed or when significant improvements in the best solution seen to date are no longer apparent.

Ways of defining *select* used in GAs include:

1. Roulette Wheel (stochastic sampling with replacement)
   This repeatedly selects a solution from P and adds it to Q with probability equal to its relative fitness. High value solutions will tend to be replicated.

2. Random
   This selects strings entirely at random from P.

3. Ranking Mechanisms
   P is ordered by fitness and Q is selected acccording to some criteria which favours those elements of P highest in the ranking. For example,

   (a) Tendency
   A sub-population is defined as an upper percentage of P ordered by fitness. The mechanism then selects randomly from P, but in such a way as to favour selections from this sub-population.

   (b) Exclusive
   This uses the same sub-population defined in the tendency mechanism, but selects solutions at random only from this sub-population.

   (c) Exponential
   Solutions in P are ordered by their values, highest values first. The mechanism then selects from this ordered list using random numbers drawn from an exponential distribution.

   (d) Fibonacci
   This selects solutions from the ordered list of solutions according to the Fibonacci sequence. Thus, the best, 2nd., 3rd., 5th., 8th., etc. is selected.

The multiset, Q, formed using *select*, represents a multiset of "fit" solutions which form the "mating" pool. The function *create* applies genetic operators to this pool to

generate the offspring. The most fundamental operator is *crossover* which is applied to two parent solutions to generate two offspring solutions. One point crossover proceeds in two stages:

Step1 A crossover site is selected, usually at random. This determines the point in each string at which all subsequent characters further along the string are swapped with characters at the same positions within the 'mate' string. For a string of length l characters, there are l -1 possible crossover points.

Step2 The strings then interchange all characters which occur after the selected crossover point. For example, using two binary strings of length six characters with a crossover point of two (shown as '|'):

11|0101
01|1001

After crossover, these strings become:

111001
010101

Multiple point crossover introduces a new parameter CP, which represents the number of crossover points. Single point crossover is then equivalent to multiple point crossover with CP set to 1. With even values of CP, the two strings are treated as rings; crossover points are then selected at random from around the ring. Odd values of CP result in a default crossing point at the start of the string being assumed.

In uniform crossover, a third, random bit string, *rand*, is generated. Let the two parent strings be $P1$ and $P2$. The two children, $C1$ and $C2$ are then constructed as follows.

$$\begin{aligned} C1_i &= P1_i \text{ if } rand_i = 1, \\ &= P2_i \text{ otherwise.} \end{aligned}$$

and

$$\begin{aligned} C2_i &= P2_i \text{ if } rand_i = 1, \\ &= P1_i \text{ otherwise.} \end{aligned}$$

Although crossover is an essential component of any GA, it is not necessarily performed between all pairs of strings in the mating pool, Q; two other genetic operators are also used. One is *replication*, which merely generates a copy of a string and the other is *mutation* in which a copy is made except for some minor random change. If the solutions are represented as bit strings, this is merely obtained by randomly resetting some bit in the string. Generally, a small degree of mutation can help to avoid a GA getting trapped in a local optimum.

Finally, we need to describe how *merge* is used in a GA to combine the old population, P, the population of solutions in the mating pool, Q and the new population, R, of solutions produced by *create*. The following mechanisms are some of those commonly exploited by the merge phase of GAs:

1. Replace all
   The new P is simply R.

2. Best fit
   The best solution in R is used to replace the worst solution in P, until either $R = \Phi$ or all solutions in P are better than those left in R.

3. New solutions
   This is the same as best fit, except that all strings in R that exist in P are discarded beforehand, since they are not new.

4. Replace
   This allows specification of a *pressure* parameter, so that only strings in R with fitness greater than that of the worst solution in P multiplied by *pressure* replace strings in P.

5. Random
   P and R are merged; strings from the new population which are not members of Q are then removed at random until the population size is equal to that of P before the merge.

There has been considerable progress in developing theories to explain why genetic algorithms work [63, 88]. Essentially, the theory establishes that the average fitness of the solution pool must slowly increase. Certainly, there is now a large number of case studies which support the efficacy of the GA approach. Moreover, it is a fairly robust technique; for many applications, the parameter settings are not nearly so critical as in either simulated annealing or tabu search. For a quick and effective way of obtaining reasonable solutions, GAs have much to offer. They have been used successfully in a wide variety of applications including the following.

- Packing and Scheduling [5, 39, 72, 76, 90].
- Graph Problems [18, 49].
- Neural Networks [7, 42, 59, 77].
- Travelling salesman [38, 40, 84, 90].

GAs may fail to yield satisfactory solutions to problems for many reasons. One common cause is that the problem representation may be inconsistent with the crossover operation. This results in the crossover operation producing offspring which no longer represent viable solutions.

This failure can be addressed by revising the problem representation. However, in situations where this is not easily achieved, an alternative crossover scheme may be devised. Unfortunately, the theory does not allow such an alternative crossover scheme and therefore such a scheme may result in an algorithm which is not, strictly speaking, genetic. However, in practice such an approach can still work satisfactorily and this highlights the need to keep a flexible approach when dealing with GAs.

The travelling salesman problem provides such an example of inconsistency between representation and crossover. Let us define a solution to the n city problem by a string of n distinct elements of $\{1, 2, \ldots, n\}$. Thus, if $n = 6$, two possible solutions might be 463125 and 261435. If these undergo crossover at the third site, the

strings 463435 and 261125 are produced, neither of which represent possible solutions. Rather than give an alternative representation, an alternative crossover scheme, partially mapped crossover (PMX) was devised. This selects two crossover points and, for each character position which these encompass, characters from both parents at that position are swapped. For each pair of characters x from parent a and y from parent b, a second pair (y from parent a and x from parent b) is swapped. For example, if the two strings shown below undergo PMX with sites 3,5 (indicated by |):

String a: 4 6 3 | 1 2 | 5
String b: 2 6 1 | 4 3 | 5

then 1 in a, 4 in b and vice versa are swapped, together with 2 in a and 3 in b and vice versa, yielding:

```
Offspring 1: 1 6 2 4 3 5
Offspring 2: 3 6 4 1 2 5
```

6 Case Study: Steiner Tree

Steiner's Problem in Graphs (SPG) is a classic combinatorial optimisation problem which involves connecting a given subset of a graph's vertices as cheaply as possible. More precisely, given a graph $G = (V, E)$ with vertices V, edges E, a cost function $c : E \rightarrow Z^+$, and a set of special vertices, $K \subseteq V$, a Steiner tree is a connected subgraph, $T = (V_T, E_T)$, such that $K \subseteq V_T \subseteq V$, and $|E_T| = |V_T| - 1$. The SPG problem is to find a Steiner tree T which minimises the cost function, $c(T) = \sum_{e \in E_T} c(e)$. Such a tree is referred to as a minimal Steiner tree. A minimal Steiner tree is not normally a minimum spanning tree on just the special vertices, it also spans some non-special vertices; the vertices $V_T \setminus K$ are referred to as Steiner vertices.

All Steiner vertices must have degree ≥ 2; it is clear that any Steiner vertex with degree one can be removed from T, resulting in a Steiner tree T' with $c(T') < c(T)$.

The vertices V_T can be partitioned into two sets, the key and non-key vertices. We define the set of key vertices $V_{key}(T)$ in Steiner tree T to be

$$V_{key}(T) = \{v \in V_T \mid v \in K \ \vee \ d_T(v) \geq 3\}$$

where $d_T(v)$ is the number of edges incident to vertex v in subgraph T. Key vertices are either special vertices or Steiner vertices acting as junctions where two or more paths meet. A key path in Steiner tree T is a simple path

$$\begin{array}{rll} p = & < v_1, v_2, \ldots, v_n > & \text{s.t.} \\ & (v_i, v_{i+1}) & \in E_T \text{ for } 1 \leq i < n, \\ & v_1, v_n & \in V_{key}(T) \text{ and} \\ & v_j & \notin V_{key}(T) \text{ for } 1 < j < n. \end{array}$$

In other words, a key path connects two key vertices via zero or more intermediate non-key vertices. The set of key paths $Q = kp(T)$ consists of all key paths in T.

In Figure 6, special vertices are shaded. The minimum Steiner tree consists of the four paths

$$Q = \{< 1, 0, 2 >, < 2, 3 >, < 2, 4 >, < 3, 6 >\}.$$

The vertices, 0 and 2, are Steiner vertices and 2 is the only key vertex $\notin K$.

All feasible Steiner trees can be uniquely represented by their set Q of key paths; the Steiner tree is trivially constructed from Q by taking the union of the paths, let $T(Q)$ denote such a tree. A minimal Steiner tree will have a key paths that are the shortest paths between its key vertices. It is straightforward to show that a Steiner tree contains at most $2|K|-2$ key vertices, and therefore at most $2|K|-3$ key paths.

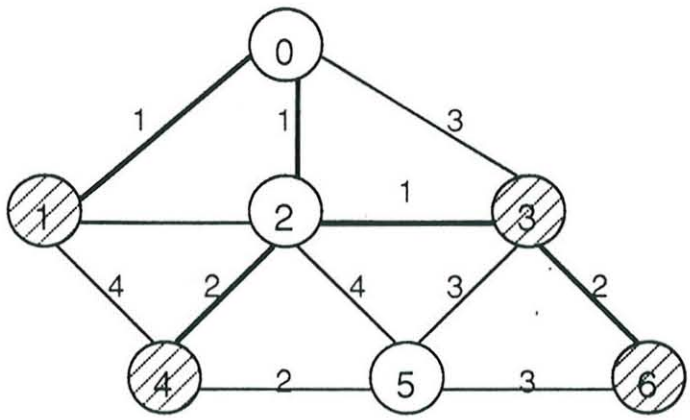

Figure 6: The Steiner problem in graphs

The Steiner problem in graphs is NP-hard [51]. Despite this result there exist special cases solvable in polynomial time. Of particular relevance in a Local Search context are those distinguished by the number of special vertices, $|K|$.

$|K| = 2$ With only two special vertices the SPG reduces to finding the shortest path between the two vertices, a problem solvable in $O(|V|^2)$ time using Dijkstra's algorithm [21], or $O(|E| \lg |V|)$ time using a modified version of the algorithm.

$|K| = 3$ The tree will consist of two paths linking the special vertices, or three paths linking the special vertices and a further intersection vertex. Given the intersection vertex, the solution is simply the union of the three shortest paths from this vertex to the special vertices. As there are a maximum of $|V|$ choices of intersection vertex, and the shortest paths can be found in $O(|E| \lg |V|)$ time, an SPG instance with $|K| = 3$ can be solved in $O(|V|(|E| \lg |V|))$ time using this technique.

$|K| = |V|$ Where all vertices are special the SPG coincides with the Minimum Spanning Tree (MST) problem, solved in $O(|E| \lg |E|)$ or better time by variants on Prim's or Kruskal's algorithms [61].

Since first formulated [41], the SPG has been the subject of a great deal of research effort. A number of exact techniques have been applied to SPG, e.g. Branch and Bound [4], Branch and Cut [14], Spanning Tree Enumeration [41], and Dynamic Programming [26]. Despite their success these techniques all suffer from exponential worst case running time. Hence there is interest in heuristic approaches, generally aiming to reduce computational effort at the expense of guaranteed optimal solutions. A number of heuristics fall into three main categories, reduction techniques, constructive algorithms, and local search variants.

Reduction methods are applied to instances of combinatorial optimisation problems in an effort to reduce problem size. SPG provides ample scope for devising methods for reducing the values $|V|$ and $|E|$. The general approach is to identify edges or vertices guaranteed not to belong to a minimal Steiner tree; eliminating these from G results in an equivalent but smaller instance. Conversely, identifying edges or vertices guaranteed to form part of a minimal Steiner tree allows a partial solution to be generated. The reduced instance will have fewer candidate solutions than the original, and hence is easier to solve. Indeed, for some small instances based on sparse graphs or with high special vertex density, applying a combination of reduction techniques may be sufficient to find optimal solutions [27].

Constructive algorithms for SPG are based on connecting vertices in K by inserting paths between subtrees. An algorithm starts with U consisting of a forest of disjoint single element trees covering K, and at each iteration selecting two trees in U and joins them via a path. After $|K| - 1$ iterations the algorithm terminates with U consisting of a single Steiner tree. One of the first such algorithms proposed was the Shortest Path heuristic (SPH) [83]. In SPH all components of U remain singleton, apart from the tree T_{vstart} containing the arbitrarily chosen starting vertex v_{start}. For each constructive step the shortest paths from T_{vstart} to all vertices in $K \setminus V_{T_{vstart}}$ are calculated using Dijkstra's algorithm, and the vertex having the shortest such path is connected to T_{vstart} by that path.

The Average Distance Heuristic, [74], improves on SPH's straightforward greedy algorithm by including a form of heuristic lookahead. At each step, a function identifies a vertex, $v \in V \setminus V_U$, such that the average distance to at least two trees in U is minimised; v is considered to be the vertex heuristically closest to the forest U. With v identified, the two vertices v_1, v_2 in V_U with the shortest paths to v are found, and the trees T_{v_1}, T_{v_2}, and vertex v connected via these shortest paths.

The Local Search approach of Dowsland [25] is based on the key path Steiner tree representation. The initial solution is generated using the SPH algorithm, or a randomised SPH variant. Local Search is applied to this solution using a neighbourhood function based on the exchange of key paths. If the solution is represented by its key paths, $Q = kp(T)$, and an arbitrary key path, p, is removed from Q, then the resulting graph consists of two disconnected trees. The trees can be optimally reconnected by considering each component as a single vertex and using Dijkstra's algorithm to find the shortest path between them. Dowsland uses $O(V^2)$ shortest path algorithm.

The removal of p and the reconnection of the two trees by the shortest path between them is described by Dowsland as a 1-opt move. Dowsland uses this 1-opt neighbourhood in a Steepest Descent Local Search algorithm. An extended key path neighbourhood is also presented, referred to as 2-opt. Two paths are removed from Q, disconnecting the tree into three sub-trees. These can be optimally reconnected, in a similar manner to the 1-opt case, by treating each sub-tree as a single special vertex forming a new SPG instance with $|K| = 3$. As we showed previously this is solvable in $O(|V|(|V| \lg |E|))$ time, although Dowsland used a $O(|V|^3)$ implementation. Testing Steepest Descent with 1-opt and 2-opt neighbourhoods on a set of 100 vertex problems showed 2-opt consistently found better solutions, but with significantly longer execution times.

Verhoeven, in [86], uses the same representation and 1-opt neighbourhood for his SPG neighbourhood search algorithm. A Random Descent algorithm is used, with initial solutions generated by the SPH algorithm. Verhoeven's implementation includes

an efficient $O(|E| \lg |V|)$ shortest path algorithm which is better than Dowsland's $O(|V|^2)$ for relatively sparse graphs. This makes it feasible to apply the algorithm to much larger SPG instances.

A Genetic Algorithm for SPG is presented in [50]. In this paper, the representation and neighbourhood function are based on the identification of Steiner vertices. A solution is represented by a bitstring of length $|V|$, with each bit i corresponding to a vertex $v_i \in V$. A '1' in position i means vertex v_i is present in the Steiner tree represented, and conversely a '0' indicates v_i is excluded. Note that to represent a valid Steiner tree it is necessary for bit i to be '1' for all $v_i \in K$, hence the bitstring identifies a set, W, of Steiner vertices. A new SPG instance $(G' = (V', E'), K')$ can be generated from $G, W,$ and K thus:

$$\begin{aligned} G' &= sg(G, K \cup W) \\ K' &= K \cup W \end{aligned}$$

where the function $G' = sg(G, \mathcal{V})$ defines the subgraph induced in graph G by the vertices $\mathcal{V}$ i.e.

$$\begin{aligned} V' &= \mathcal{V} \\ E' &= \{(v_1, v_2) \in E \mid v_1, v_2 \in \mathcal{V}\}. \end{aligned}$$

The generated SPG instance has the property that all vertices are special and hence can be solved to optimality in $O(|E| \lg |E|)$ time using Kruskal's MST algorithm. Let $T(W)$ denote the solution of the generated instance. This Steiner vertex representation does not ensure $T(W)$ is a Steiner tree for the original SPG instance. There are two possibilities for representing invalid Steiner trees. First, the subgraph G' may consist of more than one component, and if K is not spanned by a single such component then no Steiner tree exists in G'. Second, and less crucially, $T(W)$ may contain Steiner vertices of degree one.

A pool of Steiner vertex representation solutions is maintained. GA search is applied to the population: child solutions are generated by standard crossover and mutation operators. Solutions representing infeasible Steiner trees due to disconnection are priced out of the search; a large penalty is applied to the fitness function of any such solution. Solutions with Steiner vertices of degree one are tolerated on the assumption that the GA search will discourage such solutions. Good quality results are presented for the Beasley B set but the algorithm is rather less successful on the larger instances (C and D sets).

Another genetic algorithm with a vertex based representation is presented in [28]. Although the bitstring used is similar to [50], the relationship between a bitstring and resulting tree is different. The vertices specified by the bitstring are used as parameters for the heuristic algorithm of [53]. Good quality results are presented for the Beasley C, D, and E sets. Execution time is several hours for the larger instances, despite all instances being pre-processed with a SPG reduction algorithm. Furthermore, the technique requires calculation and storage of the all pairs shortest paths, resulting in a $O(|V|^2)$ memory requirement which limits scalability.

A simulated annealing algorithm for a related problem, the directed Steiner problem on networks, is presented in [67]. A similar representation and cost function to [50] is used. The neighbourhood function is based on adding or removing single vertices from the set of Steiner vertices. Results are presented for small instances with at most 80 vertices.

Currently, the best heuristic algorithm for SPG uses both the key path representation and the Steiner vertex representation. The algorithm is essentially a simulated annealing algorithm which regularly changes topology by changing representation and neighbourhood function. Details are given in [89]; this approach can solve successfully all the Beasley instances and, in most cases, solutions are found in under 10 minutes.

7 Case Study: Data Mining

Let us assume we have a relational database, D, of d records in the form of a flat file and that all fields have known values for each $r \in D$. D consists of a set of records specifying values for the sequence of attributes

$$A =< A_1, A_2, \ldots, A_n >$$

where A_i is defined over the over the domain Dom_i, $1 \leq i \leq n$.

We are asked to find a rule of the form

$$\alpha \Rightarrow \beta$$

which appears to hold for some records in this database.

Let r denote a particular record in the database and γ denote some predicate defined in terms of the n attributes. $\gamma(r)$ will then be true if the record r satisfies the predicate γ.

A rule expressed as above represents knowledge extracted from D and the process of determining such rules is often known as *Knowledge Discovery in Databases*. For a good introduction to this subject see [45, 56].

Associated with any rule

$$\alpha \Rightarrow \beta$$

are three sets of records,

$$\begin{aligned} A &= \{r \mid \alpha(r)\}, \\ B &= \{r \mid \beta(r)\}, \\ \text{and} & \\ C &= \{r \mid \alpha(r) \wedge \beta(r)\} \\ &= A \cap B. \end{aligned}$$

We can then define the integer values a, b and c by

$$a = |A|,\ b = |B| \text{ and } c = |C|.$$

Figure 7 illustrates the situation. Remembering that d denotes the size of the database, one can define various ratios between these values which measure properties of the rule. We will use

$App(\alpha \Rightarrow \beta) = \frac{a}{d}$	APPLICABILITY of the rule, and
$Acc(\alpha \Rightarrow \beta) = \frac{c}{a}$	ACCURACY of the rule.

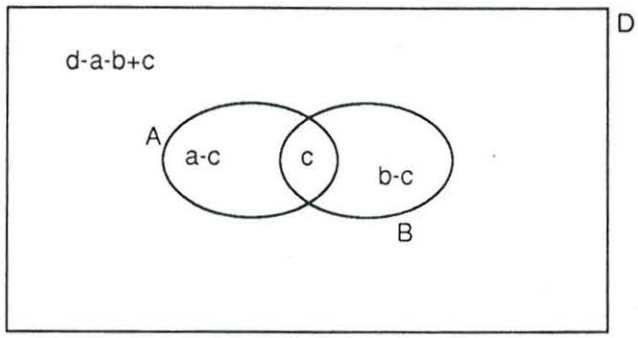

Figure 7: Venn diagram for a rule in a database of defined records.

In general, we will be searching for one or more rules which meet certain predefined criteria. These criteria are designed to determine how well the rule describes (some part of) the database. So, for example, we might be required to search for a rule that has at least 90% accuracy and has as high applicability as possible.

In order to reduce the size of the search space, it might be necessary to restrict the format of the rules we seek. For example, α, the precondition of the rule might be restricted to be a conjunction of the form

$$(A_{\lambda_1} \in v_{\lambda_1}) \wedge (A_{\lambda_2} \in v_{\lambda_2}) \wedge \ldots \wedge (A_{\lambda_i} \in v_{\lambda_i})$$

where $1 \leq \lambda_1 < \lambda_2 < \ldots < \lambda_i \leq n$ and where

$$v_{\lambda_1} \subseteq Dom_{\lambda_1}, v_{\lambda_2} \subseteq Dom_{\lambda_2}, \ldots, v_{\lambda_i} \subseteq Dom_{\lambda_i}.$$

The predictive part of the rule, β, might then be similarly restricted take the form of

$$(A_j \in v_j),$$

for some $j \notin \{\lambda_1, \lambda_2, \ldots, \lambda_i\}$ and $v_j \subseteq Dom_j$.

Assuming that the domains are meaningfully ordered, such rules can than be represented as an array of suitably encoded upper and lower limits (one pair for each of the input fields and one pair for the output field).

Once the format of the rule is determined together with its representation, we need to define a fitness (or value) function. Given a rule, this function determines how well that rule meets the given criteria. Several suggestions for defining these criteria have been proposed in the literature, see for example [45, 75]. These can be quite complex involving fuzzy sets, information gain and rule simplicity measures as well as expressions in a, b, c and d. Nevertheless, there have been cases where very simple value functions have proved remarkably successful. In [75], a case study was described where the predictive field was a preset Boolean value and the value function was just

$$\lambda c - a, \ \lambda \in R$$

yet useful rules were extracted.

Once this fitness value is determined, data mining then becomes an optimisation problem – we seek the "best" rule, according to our measure, within the space of all rules. We have used simulated annealing and genetic algorithms to find such rules and our recent research is reported in [19].

8 Hybridisation

In this section, we will suggest a unifying framework for the various search strategies that we have considered. Other attempts to do this have been presented in [29, 71, 73].

Our key observation is that it is possible to view GAs as yet another type of neighbourhood search. Rather than view *neighbour* as a function $U \rightarrow 2^U$, we can extend the concept and define

$$neighbour_2 : U \times U \rightarrow 2^U.$$

The crossover operation merely defines a neighbourhood of *two* solutions in U, taken together.

Given a submultiset of solutions, $Q \subset P$, we can define *create* so that it generates some submultiset of $N_1 \cup N_2$ where

$$\begin{aligned} N_1 &= \bigcup\{neighbour(u) \mid u \in Q\} \cdot \\ \text{and} & \\ N_2 &= \bigcup\{neighbour_2(u, v) \mid u, v \in Q\}. \end{aligned}$$

N_1 corresponds to mutants and N_2 to children. The $neighbour_2$ function may not be pure crossover but, may be defined in a more complex manner.

Viewed in this way, the GA paradigm is merely a simple extension of neighbourhood search. Currently, most GAs only randomly select at most one solution from $neighbour_2(u, v)$ for any $u, v \in Q$. It may well be worthwhile investigating variants of this technique whereby the fitter children are more likely to be generated. This could be achieved by either generating more children in a single mating cycle and then culling weaker offspring and/or by restricting crossover points according to some (possibly dynamically changing) rules.

In simulated annealing, the essential idea is to allow weak mutants to survive providing it is warm enough. GAs also have this facility of allowing some weak members to survive in the solution pool but always have mechanisms for favouring fitter solutions. The GA paradigm might be strengthened with the inclusion of a temperature parameter or at least some dynamic control of its parameters. The GAmeter package allows such control but only by human intervention.

The GA paradigm can also be considerably enhanced by learning some of the lessons of tabu search. By maintaining a dynamic list of rules, both N_1 and N_2 can be pruned and/or ranked by desirability. Rather than simply use *value* as the value measure, we can also use such a ranking as an additional or replacement measure. The use of such expert guidance need not be restricted to *create*. We must have definitions for each of the functions, *initialise*, *finish*, *select*, *neighbour*, *create* and *merge*; we can envisage algorithms where these definitions are quite sophisticated. Tabu search suggests that the inclusion of an expert system to define some of these definitions may be appropriate. Moreover, if such an expert system is given as a set of rules, these can be updated during the running of the program. This could exploit the temperature coefficient used by simulated annealing.

The general search method in nearly all of its many variations has considerable scope for parallelism [11, 48, 57, 62, 70, 93]. The evaluation of the solutions, generation, evaluation and selection of neighbours together with much of the merging can be performed in parallel.

9 UEA Toolkits

At UEA, we have developed a GA toolkit, GAmeter [48], which supports all of the above options for *select*, *create* and *merge* as well as some others. The toolkit can run on UNIX or Macintosh workstations and there is a parallel version running on a Meiko transputer rack and a distributed version using PVM on a network of UNIX workstations. It enables very fast implementation of GAs and allows easy experimentation with parameter settings.

Similarly, we have developed a toolkit, SAMSON, supporting Simulated Annealing. It supports a wide range of SA options and both UNIX and Macintosh versions are available. The distributed version is about to be released for trial.

The toolkit for Tabu Search, TAbasco, is less developed but a preliminary version is being used in our department. We intend to develop this further and to implement a distributed version. Our next step is to build an adaptive, distributed search engine.

GAmeter and SAMSON are available free to academic researchers and copies can be obtained by emailing me at vjrs@sys.uea.ac.uk. Further details of the toolkits and of the research activities of the Mathematical Algorithms Group is available on our web page

(http://www.sys.uea.ac.uk/Research/ResGroups/MAG/).

References

[1] D. Abramson. Constructing school timetables using simulated annealing: sequential and parallel algorithms. *Man. Sci.*, 37:98–113, 1991.

[2] B. Adenso-Diaz. Restricted neighborhood in the tabu search for the flowshop problem. *EJOR*, 62(1):27–37, 1992.

[3] C. A. Anderson, K. F. Jones, M. Parker, and J. Ryan. Path assignment for call routing: an application of tabu search. *Annals of Ops. Res.*, 41:301–312, 1993.

[4] J.E. Beasley. An sst-based algorithm for the Steiner problem in graphs. *Networks*, 19:1–16, 1989.

[5] S. Beaty. *Instruction scheduling using genetic algorithms: PhD Thesis.* Colorado State University, 1991.

[6] E. Bonomi and J-L. Lutton. The n-city travelling salesman problem: Statistical mechanics and the metropolis algorithm. *SIAM Review*, 26:551–568, 1984.

[7] S. Bornholdt and D. Graudenz. General asymmetric neural networks and structure design by genetic algorithms. *Neural Networks*, 5:327–334, 1992.

[8] P. Brandimarte, R. Conterno, and P. Laface. FMS production scheduling by simulated annealing. *Proceedings of 3rd International Conference on Simulation in Manufacturing*, pages 235–245, 1987.

[9] M. J. Brusco and L. W. Jacobs. A simulated annealing approach to the solution of flexible labour scheduling problems. *JORS*, 44:1191–1200, 1993.

[10] V. Cerny. A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *J. of Optimization Theory and Applic.*, 45:41–55, 1985.

[11] J. Chakrapani and J. Skorin-Kapov. Massively parallel tabu search for the quadratic assignment problem. *Annals of O.R.*, 41:327–342, 1993.

[12] M. Chams, A. Hertz, and D. deWerra. Some experiments with simulated annealing for colouring graphs. *EJOR*, 32:260–266, 1987.

[13] P. Chardaire, J.L. Lutton, and A. Sutter. Thermostatistical persistency: A powerful improving concept for Simulated Annealing algorithms. *European Journal of OR*, 86:565–579, 1995.

[14] S. Chopra, E.R. Gorres, and M.R. Rao. Solving the Steiner tree problem on a graph using branch and cut. *ORSA Journal on Computing*, 4(3):320–335, 1992.

[15] D. T. Connolly. An improved annealing scheme for the qap. *EJOR*, 46:93–100, 1990.

[16] D. Costa. On the use of some known methods for T-colorings of graphs. *Annals of OR*, 41:343–358, 1993.

[17] R. L. Daniels and J. B. Mazzola. A tabu search heuristic for the flexible-resource flow shop scheduling problem. *Annals of Ops. Res.*, 41:207–230, 1993.

[18] L. Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[19] B. de la Iglesia, J.C.W. Debuse, and V.J. Rayward-Smith. Discovering knowledge in commercial databases using modern heuristic techniques. In E. Simoudis, J. Han, and U. Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 44–49. AAAI Publications, 1996.

[20] M. DellAmico and M. Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of OR*, 41:231–252, 1993.

[21] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Math.*, 1:269–271, 1959. shortest path alg.

[22] W. Domschke, P. Frost, and S. Voss. Tabu search techniques for the quadratic semi-assignment problem. In F. Fandel, T. Gulledge, and A. Jones, editors, *New Directions for Operations Research in Manufacturing*, pages 389–405. Springer, 1991.

[23] K. A. Dowsland. Hill-climbing, simulated annealing, and the Steiner problem in graphs. *Eng. Opt*, 17:91–107, 1991.

[24] K. A. Dowsland. Simulated annealing. In C. R. Reeves, editor, *Modern Heuristic Techniques*, chapter 2. Blackwell Scientific, 1993.

[25] Kathryn A. Dowsland. Hill-climbing, Simulated Annealing and the Steiner problem in graphs. *Engineering Optimisation*, 17:91–107, 1991.

[26] S.E. Dreyfus and R.A. Wagner. The Steiner problem in graphs. *Networks*, 1:195–207, 1971.

[27] C.W. Duin and A. Volgenant. Reduction tests for the Steiner problem in graphs. *Networks*, 19:549–567, 1989.

[28] H. Esbensen. Computing near-optimal solutions to the Steiner problem in a graph using a genetic algorithm. *Networks*, 26:173–185, 1995.

[29] B. L. Fox. Integrating and accelerating tabu search, simulated annealing, and genetic algorithms. *Annals of OR*, 41:47–68, 1993.

[30] A. Freville and G. Plateau. Hard 0-1 multiknapsack test problems for size reduction methods. *Investigacion Operativa*, 1:251–270, 1990.

[31] C. Friden, A. Hertz, and D. deWerra. Stabulus: a technique for finding stable sets in large graphs with tabu search. *Computing*, 42:35–44, 1989.

[32] F. Glover. Tabu search, part 1. *ORSA J. Comp.*, 1:190–206, 1989.

[33] F. Glover and M. Laguna. Tabu search. In C. R. Reeves, editor, *Modern Heuristic Techniques*, chapter 3. Blackwell Scientific, 1993.

[34] F. Glover and C. McMillan. The general employee scheduling problem: an integration of management science and artificial intelligence. *Computers & Ops. Res.*, 15:563–593, 1986.

[35] F. Glover, C. McMillan, and B. Novick. Interactive decision software and computer graphics for architectural and space planning. *Annals of Ops. Res.*, 5:557–573, 1985.

[36] F. Glover, E. Taillard, and D. deWerra. A user's guide to tabu search. *Annals of OR*, 41:3–28, 1993.

[37] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, Mass., 1989.

[38] D. E. Goldberg and R. Lingle. Alleles, loci and the travelling salesman problem. In J. J. Grefenstette, editor, *Proc. of an International Conf. on Genetic Algorithms and their applications*, pages 154–159, Hillsdale, New York, 1985. Lawrence Erlbaum.

[39] D. E. Goldberg and R. E. Smith. Nonstationary function optimization using genetic algorithms with dominance and diploidy. In L. Davis, editor, *Handbook of Genetic Algorithms*, pages 59–68, New York, 1991. Rheinhold.

[40] J. J. Grefenstette. Incorporating problem-specific knowledge into genetic algorithms. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, pages 42–60, Los Altos, CA, 1987. Morgan Kauuffman.

[41] S.L. Hakimi. Steiner's problem in graphs and its applications. *Networks*, 1:113–133, 1971.

[42] S. A. Harp and T. Samad. Genetic synthesis of neural network architecture. In L. Davis, editor, *Handbook of Genetic Algorithms*, pages 202–221, New York, 1991. Rheinhold.

[43] A. Hertz and D. deWerra. Using tabu search techniques for graph coloring. *Computing*, 29:345–351, 1987.

[44] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[45] M. Holsheimer and A. Siebes. Data mining: The search for knowledge in databases. *Technical report CS-R9406, CWI Amsterdam*, 1994.

[46] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation. *Opns. Res.*, 37:865–892, 1989.

[47] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation; part ii, graph coloring and number partitioning. *Opns. Res.*, 39:378–406, 1991.

[48] A. Kapsalis, V. J. Rayward-Smith, and G. D. Smith. Fast sequential and parallel implementation of genetic algorithms using the gameter toolkit. In R. F. Albrecht, C. R. Reeves, and N. C. Steele, editors, *Proc. International Conf. on Neural Networks and Genetic Algorithms*. Springer-Verlag, 1993.

[49] A. Kapsalis, V. J. Rayward-Smith, and G. D. Smith. Solving the graphical Steiner tree problem using genetic algorithms. *J. Oper. Res. Soc.*, 44(4):397–406, 1993.

[50] A. Kapsalis, V.J. Rayward-Smith, and G.D. Smith. Solving the graphical Steiner tree problem using genetic algorithms. *J. Oper. Res. Soc.*, 44:397–406, 1993.

[51] R. M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[52] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[53] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Informatica*, 15:141–145, 1981.

[54] S. A. Kravitz and R. A. Rutenbar. Placement by simulated annealing on a multiprocessor. *IEEE Trans. on Computer Aided Design*, CAD-6:534–549, 1987.

[55] M. Laguna, J. W. Barnes, and F. Glover. Tabu search methods for a single machine scheduling problem. *J. of Intelligent Manufacturing*, 2:63–74, 1991.

[56] P. R. Limb and G. J. Meggs. Data mining - tools and techniques. *BT Technol J*, 12, 1994.

[57] D. Macfarlane and I. East. An investigation of several parallel genetic algorithms. In *Proc. 12th Occam User group, Technical Meeting*, pages 60–67, 1990.

[58] M. Malek, M. Guruswamy, M. Pandya, and H. Owens. Serial and parallel simulated annealing and tabu search algorithms for the travelling salesman problem. *Annals of Ops. Res.*, 21:59–84, 1989.

[59] S. J. Marshall and R. F. Harrison. Optimization and training of feedforward neural networks by genetic algorithms. In *Proc. 2nd IEE International Conference on Artificial Neural Networks*, pages 39–43, 1991.

[60] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculation by fast computing machines. *J. of Chem. Phys.*, 21:1087–1091, 1953.

[61] B.M.E. Moret and H.D. Shapiro. *Algorithms from P to NP. Volume I: Design and Efficiency.* Benjamin-Cummings, Menlo Park, CA, 1991.

[62] H. Muhlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. In R. K. Belew and L. B. Booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*, San Mateo, CA, 1991. Morgan Kaufmann.

[63] A. Nix and M. Vose. Modelling genetic algorithms with Markov chains. *Annals of Maths. and AI*, pages 79–88, 1992.

[64] F. A. Ogbu and D. K. Smith. The application of the simulated annealing algorithm to the solution of the $n/m/c_{max}$ flowshop problem. *Computers and Ops. Res.*, 17:243–253, 1990.

[65] F. A. Ogbu and D. K. Smith. Simulated annealing for the permutation flow-shop problem. *Omega*, 19:65–67, 1991.

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

[67] L. Osborne and B. Gillet. A comparison of two Simulated Annealing algorithms applied to the directed Steiner problem on networks. *ORSA Journal on Computing*, 3(3):213–225, 1991.

[68] I. H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41:421–451, 1993.

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

[70] C. B. Petty, M. R. Leuze, and J. J. Grefenstette. A parallel genetic algorithm. In *Genetic algorithms and their applications: Proc 2nd Int. Conf. Genetic Algorithms*, 1987.

[71] M. Pirlot. General local search heuristics in combinatorial optimization: a tutorial. *Belgian Journal of OR, Statistics and Computer Science*, 32:7–67, 1993.

[72] P. Prosser. A hybrid genetic algorithm for pallet loading. In *Proc. 8th European Conference on Artificial Intelligence*. Pitman, London, 1988.

[73] V. J. Rayward-Smith. A unified approach to tabu search, simulated annealing and genetic algorithms. In *Adaptive Computing and Information Processing*, pages 55–78. UNICOM Seminars Ltd., 1994.

[74] V.J. Rayward-Smith and A. Clare. On finding Steiner vertices. *Networks*, 16:283–294, 1986.

[75] V.J. Rayward-Smith, J.C.W. Debuse, and B. de la Iglesia. Using a genetic algorithm to data mine in the financial services sector. In A. Macintosh and C. Cooper, editors, *Applications and Innovations in Expert Systems III*. SGES Publications, 1995.

[76] C. R. Reeves. A genetic algorithm approach to stochastic flowshop sequencing. In *Proc. IEE Colloquium on Genetic Algorithms for Control and Systems Engineering*. IEE, London, 1992.

[77] C. R. Reeves and N. C. Steele. Problem-solving by simulated genetic processes: a review and application to neural networks. In *Proc 10th IASTED Symp. .on Applied Informatics*, Anaheim, CA, 1992. ACTA Press.

[78] Y. Rossier, M. Troyon, and T. M. Liebling. Probabilistic exchange algorithms and euclidean travelling salesman problems. *OR Spektrum*, 8:151–164, 1986.

[79] F. Semet and E. Taillard. Solving real-life routing problems efficiently using taboo search. *Annals of Ops. Res.*, 41:469–488, 1993.

[80] J. Skorin-Kapov. Tabu search applied to the quadratic assignment problem. *ORSA J. on Computing*, 2:33–45, 1990.

[81] E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *EJOR*, 47:65–74, 1990.

[82] E. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17:443–455, 1991.

[83] H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Mathematica Japonica*, 24:573–577, 1980.

[84] S. R. Thangiah, I. H. Osman, R. Vinayagamoorthy, and T. Sun. Algorithms for vehicle routing with time deadlines. *American Journal of Mathematical and Management Sciences*, 13, 1994.

[85] A. J. Vakharia and Y. L. Chang. A simulated annealing approach to scheduling a manufacturing cell. *NRL*, 37:559–577, 1990.

[86] M.G.A. Verhoeven, E.H.L. Aarts, and M.E.M. Severens. Local search for the Steiner problem in graphs. *Proceedings of Applied Decision Technologies*, pages 345–351, 1995.

[87] A. Vernekar, G. Anandalingam, and C. Dorny. Optimization of resource location in hierarchical computer networks. *Computers & Ops. Res.*, 17:375–388, 1990.

[88] S. Vose. Modeling simple genetic algorithms. In D. Whitley, editor, *Foundations of Genetic Algorithms 2*. Morgan Kaufmann, 1993.

[89] A.S.C. Wade and V.J. Rayward-Smith. Effective local search techniques for the steiner tree problem. *submitted for publication.*

[90] D. Whitley, T. Starkweather, and D. Shaner. The traveling salesman and sequence scheduling: quality solutions using genetic edge recombination. In L. Davis, editor, *Handbook of Genetic Algorithms*, pages 350–372, New York, 1991. Rheinhold.

[91] M. Widmer. Job shop scheduling with tooling constraints: a tabu search approach. *JORS*, 42:75–82, 1991.

[92] M. Widmer and A. Hertz. A new heuristic for the flow shop sequencing problem. *EJOR*, 41:186–193, 1989.

[93] E. E. Witte, R. D. Chamberlain, and M. A. Franklin. Parallel simulated annealing using speculative computation. *IEEE Trans. Par. Dist. Sys.*, 2(4):483–494, 1991.

DISCUSSION

Rapporteur: Dr Rogério de Lemos

Lecture One

During the talk Professor Randell enquired whether during a search that starts at multiple starting points there was a danger of overlapping while searching in the same area. Professor Rayward-Smith replied that that was the kind of problems raised in the selection and replacement; the wish was either to diversify or to intensify the search around a certain area, because all the solutions are bound to have the same features.

Professor Randell made the remark that if the specific modifications in the searching techniques discussed in the talk were likely to be more appropriate for the various particular types of problems, then the difficulty would become the identification of the different types of problems. Professor Rayward-Smith agreed with the statement and added that it was hard to try and one had to be an expert in the field. Professor Randell continued by asking whether instead of selecting the appropriate techniques for selected problems one could have general rules about problems. Professor Rayward-Smith replied that although it was possible to diversify and intensify, it was not so easy to apply general rules to unseen problems.

Professor Henderson asked if genetic algorithms could work without mutation. Professor Rayward-Smith answered that it would work occasionally, but in practice in all the problems that he had encountered the genetic algorithms work much better with a bit of mutation. Professor Hesselink made the comment that if the initial pool was too small then there was the need for mutation.

Professor Randell asked if there was in the community a satisfactory set of tests cases. Professor Rayward-Smith replied that the community had set up libraries of test cases, however these represented very classic academic problems which were quite different from the single problems that were found in the real world.

After the talk Professor Burns asked the speaker to comment about neural nets. Professor Rayward-Smith said that neural networks were effective in data mining, in other words, to clean up the data, otherwise there was tremendous hostility on its use from industry because they were essentially a black box technology. On the other hand, genetic algorithms were easier to explain and to make sense out of them.

Dr Larcombe made the comment that in her opinion while heuristics were a violence, genetic algorithms were a mindless violence because while one could prove something with heuristics, with genetic algorithms it was difficult to obtain a relevant proof. Professor Rayward-Smith agreed, and said that it was a fair comment to be made.

Professor Nievergelt made the comment that if one was able to abstract from a specific problem, then there was not any reason to say that a particular algorithm was better than the other, because in search all cleverness was related with the identification of a specific problem and its mathematical structure. Professor Rayward-Smith replied that there were principles that one could learn at every application of a genetic algorithm, however at the end, it really depended on the understanding that one had of the problem.

Dr Bird asked whether from the results presented one could conclude that one class of techniques was much broader than another class, for instance, could one say that simulated annealing could be formally represented by genetic algorithms. Professor Rayward-Smith replied that simulated annealing could be considered a subclass of tabu, depending on the rule set, however he would prefer to see the techniques as intercepting variants of local search. Instead of fitting one technique into another and finding a general one, he would prefer to generalise local search to encompass all the techniques.

Lecture Two

During the talk, while commenting on the difficulties of convincing industrial managers about the efficiency and accuracy in the utilisation of certain rules, Professor Randell enquired whether it would not be easier if the managers could convince themselves about a rule that they did not follow beforehand, if that rule had been applied in retrospect. Professor Rayward-Smith replied that he did not know of any rule introduced to these managers that they had to convince themselves that was right beforehand; he continued by saying that the reason managers were not interested in neural nets was because they would not be able to justify to their senior managers about the accuracy and efficiency of neural nets in taking dramatic decisions.

After the talk, Dr Andersson asked the speaker to comment on the relation between thinking and hacking, in the sense that in this field there is no need to prove what is done, except through benchmarks. Professor Rayward-Smith answered that there were standard techniques that hackers use, moreover there was an early training element which provided the educational value. He continued by saying that there were some underlying principles in the material he had presented that could be used in an algorithm course; these principles would not be adequate for a course at the undergraduate level, which should deal with classical algorithms, but perhaps at the postgraduate level. He also emphasised that there was a great demand in industry for these skills, and universities should be providing the technical training.

Dr Bird asked the speaker to make a broad distinction between genetic algorithms and neural nets. Professor Rayward-Smith answered that essentially genetic algorithms come up with a rule which provided the basis for understanding them, however neural nets are essentially black boxes from which it is difficult to extract a rule. Dr Bird also asked whether in neural nets a rule could be an output. Professor Rayward-Smith replied that a rule could be considered an output if there were enough outputs to cover all possible settings.

Professor Randell felt that several of the speakers had made the point of bringing together what he thought to be separate topics, and asked whether the belief in the communality and the belief in the potential value of hybridisation of different techniques was something that was generally accepted or it was restricted in the choice of speakers. Professor Rayward-Smith answered that there was already a fair proportion of hybrid algorithms.

Professor Randell also commented that he had heard many people giving talks on data mining, and these talks usually appeared to him to have little in common because of the different techniques that were applied in different sorts of problems, he wondered whether the data mining community was really a community. Professor Rayward-Smith replied that there were a lot of issues in the data mining field, and that although there were a lot of toolkits for the individual bits, there was not really an adequate toolkit that could deal with all the different issues. Dr Andersson agreed with the speaker, and added that in his opinion the problem in these communities was essentially the lack of common benchmarks which would allow comparison of the different approaches, and this was sometimes frustrating. Professor Rayward-Smith made the remark that the area was growing fast, and there was an increasing number of academics who were getting involved.