ISSN 1744-8050

A survey of AI-based meta-heuristics for dealing with local optima in local search

Technical Report Series
Report Number CSM-416
September 2004

Patrick Mills, Edward Tsang, Qingfu Zhang and John Ford
{ millph, edward, qzhang, fordj }@essex.ac.uk
http://cswww.essex.ac.uk/CSP/
Department of Computer Science,
University of Essex,
Colchester CO4 3SQ.
Telephone: +44 (0)1206 872774

Abstract

Meta-heuristics are methods that sit on top of local search algorithms. They perform the function of avoiding or escaping a local optimum and/or premature convergence. The aim of this paper is to survey, compare and contrast meta-heuristics for local search. First, we present the technique of local search (or hill climbing as it is sometimes known). We then present a table displaying the attributes of all the different meta-heuristics. After this, we give a short description and discussion of each meta-heuristic with pseudo code. Finally, we describe why, in general, these techniques work and present some ideas of what is needed from the next generation of meta-heuristics.

1 Introduction

Sitting on top of local search, meta-heuristics is a powerful technique which has been applied to many search and optimisation problems in artificial intelligence. For example, it has been successfully applied to: satisfiability (SAT) (Selman et al. 1992), planning (Kautz & Selman, 1996), constraint satisfaction (CSP) (Minton et al. 1992, Tsang, 2002), the travelling salesman problem (TSP) (Voudouris and Tsang, 1998), the quadratic assignment problem (QAP) (Mills et al. 2003), the vehicle routing problem (VRP) (Bent and Van Hentenryck, 2004) and dynamic workforce scheduling (Lesaint et al. 2003) to name but a few. Outside these problems, meta-heuristics such as Simulated Annealing have also been used as an alternative to Genetic Programming (Koza, 1992), as in O'Reilly and Oppacher (1994). Meta-heuristics are therefore an extremely important sub-discipline of Artificial Intelligence, with hundreds of papers on the topic spread over the last decade of research alone. The aim of this paper is to survey, compare and contrast meta-heuristics for local search.

1.1 The structure of this paper

This paper is broken down as follows. First we describe the concept of local search. Second, we describe what a meta-heuristic is and give a table providing features of the meta-heuristics described in this paper. Finally, we conclude by describing the features of the types of problems that meta-heuristics are good at solving, and the basic mechanisms used to exploit these.

2 Local Search

Local search algorithms work by starting with a solution (usually randomly or heuristically generated) representing some possible configuration (for example, a permutation of cities representing a "tour" in the TSP or a set of boolean variables in the SAT problem) and then making small changes to that solution which decrease[1] the cost of the configuration. A local search algorithm consist of a *neighbourhood function* for generating the set of neighbouring solutions *N(x)* of a solution *x, a cost function* (for example, the number of violated constraints as in Minton et al. 1992) for evaluating the cost or fitness *f(x)* of a solution *x* and some heuristic for choosing between solutions (for example, choosing the best solution with respect to the cost (best-improvement)). However, local search algorithms have a drawback – namely, that often, after a few *"moves"* (small changes to solutions) to neighbouring solutions, the cost function *f(x)* can no longer be reduced and the algorithm becomes stuck in what is known as a *local optimum*.

3 Meta-heuristics

Meta-heuristics are special heuristics that are designed to control heuristics like local search to enable them to either avoid or escape from the local optima described in the previous subsection. Many such meta-heuristics exist, each with many variations on the basic theme, and, due to this, we restrict our attention to the main ones. In Table 1, we show an approximate map of current meta-heuristic research (we recognise that this is by no means a complete picture or a unified view of current research, but it provides us with a basis to compare and contrast different meta-heuristics).

We split algorithms into five main classes[2]: those using some kind of random (usually non-improving) moves to randomly move out of and escape or avoid local minima, those using populations of solutions either for the purpose of restarting or for searching for multiple solutions in parallel (rather than just concentrating on a single solution), neighbourhood-modification based meta-heuristics and those using

[1] We assume, throughout this paper, that the problem being solved is represented by the minimisation of a certain cost function.

ISSN 1744-8050

penalties or weights to modify the objective function so that a local minimum can be escaped by increasing the cost of solution attributes within that local minimum. In addition to this, it is also worth distinguishing which heuristics use some form memory (apart from using a population of solutions) either long or short-term and those using some form of statistical probability model. Hence, we add three extra columns to Table 1 to represent these attributes of meta-heuristics.

Algorithm name	***Population Based***	***Random Moves***	***Probability Model***	***Neighbourhood Pruning***	***Penalty/Weight***	***Short term Memory***	***Long term Memory***
GSAT *(Selman et al., 1992)*		Y					
WalkSAT *(Selman et al., 1994)*		Y		Y			
Weighted GSAT *(Frank, 1996, 1997)*		Y			Y	Y	Y
DLM *(Shang and Wah, 1998)*					Y		Y
GENET *(Tsang & Wang, 1992*, *Davenport, 1997)*					Y		Y
Guided Local Search (GLS) (*Voudouris, 1997)*					Y		Y
Extended Guided Local Search (EGLS) (*Mills, 2002)*		Y			Y		Y
Tabu Search (TS) *Glover (1989, 1990)*				Y		Y	Y
Reactive Tabu Search (ReTS) *(Battiti and Tecchiolli, 1994)*		Y		Y		Y	
Robust Tabu Search (RTS) *(Taillard, 1991)*				Y		Y	Y
Iterated Robust Tabu Search (IRTS) (*Smyth et al. 2003)*		Y		Y			Y
Fast Local Search (FLS) / "Don't look bits" *(Voudouris,1997)*				Y		Y	
Variable Neighbourhood Search (VNS) *(Mladenovic & Hansen, 1997)*				Y			
Iterated Local Search (ILS) *(Stützle, 1999)*		Y					
Simulated Annealing (SA) *(Kirkpatrick et al. 1983)*		Y					
Simulated Jumping (SJ) *(Amin, 1999)*		Y					
GRASP *(Feo & Resende, 1995)*		Y					
Memetic Algorithms (MA) *(Moscato, 1989)*	Y						
Genetic Hybrid (GH) *(Fleurent and Ferland, 1994)*	Y			Y			
Estimation of Distribution Algorithms (EDAs) *(Zhang et al. 2003a,b)*	Y		Y			Y	Y
Scatter Search (SS) *(Cung et al. 1997)*	Y						
Path Relinking (PR) *(Glover, Laguna & Martí, 2000)*	Y						
Ant Colony Optimization (ACO) *(Gambardella et al. 1999)*	Y	Y			Y	Y	Y

Table 1: An approximate snapshot of A.I. based meta-heuristics for local search

[2] Of course, it should be noted, that many of meta-heuristics combine several attributes to produce a more effective overall heuristic.

4 Random move based meta-heuristics

In this section, we describe meta-heuristics that escape local optima by adding some form of randomness or noise to the search process when deciding what move to select from the neighbourhood of moves available.

4.1 Simulated Annealing (SA)

```
SimulatedAnnealing(StartTemperature, AnnealingSchedule())
{
      x* = x = GenerateInitialSolution()
      T = StartTemperature
      i = 0
      do
      {
            Pick a neighbour y from the neighbourhood N(x) at random
            $\Delta$f = f(y) - f(x)
            if ($\Delta$f < 0) x = y
            else
            {
                  r = random number in range [0,1]
                  if (r < $e^{-\Delta f/T}$) x = y      //Accept the change
            }
            i = i + 1
            T = AnnealingSchedule(T,i) //Reduce T according to some
                                       //Annealing Schedule
            if (f(x) < f(x*)) x* = x
      }
      while not termination condition
}
```

Figure 1: Pseudo code for Simulated Annealing

Simulated Annealing (Metropolis et al. 1956, Kirkpatrick et al. 1983) is a meta-heuristic used to navigate through the space of solutions containing many local minima and has been applied to many combinatorial optimisation problems. The main idea behind Simulated Annealing is an analogy with the way in which liquids freeze and crystallize. When liquids are at a high temperature their molecules can move freely in relation to each other. As the liquid's temperature is lowered, this freedom of movement is lost and the liquid begins to solidify. If the liquid is cooled slowly enough, the molecules may become arranged in a crystalline structure. The molecules making up the crystalline structure will be in a minimum energy state. If the liquid is cooled very rapidly it does not form such a crystalline structure, but instead forms a solid whose molecules will not be in a minimum energy state. The fundamental idea of Simulated Annealing is therefore that the moves made by an iterative improvement algorithm are like the re-arrangements of the molecules in a liquid that occur as it is cooled and that the energy of those molecules corresponds to the cost function which is being optimised by the iterative improvement algorithm. Thus, the simulated annealing algorithm aims to achieve a global optimum by slowly converging to a final solution, making downwards moves with occasional "upwards" moves (the probability of these occurring decreasing with the "temperature") and thus hopefully ending up in a global optimum. This is in contrast to the greedy approach of only considering the move which results in the largest possible decrease (if minimising) in the objective function, which resembles a rapid cooling of a liquid to a solid, and thus according to the hypothesis, resulting in a local optimum rather than a global optimum.

Figure 1 shows pseudo code for Simulated Annealing. The algorithm begins by generating an initial start point (usually at random) and setting the temperature to a suitably high value (this must be determined by experimentation). The algorithm then iteratively chooses a neighbouring solution to the current solution and evaluates the change in the cost from the current solution. If the change in the cost is negative (i.e. the neighbouring solution is better) then the move to the neighbouring solution is made. Otherwise, the move is made with probability $e^{-\Delta f/T}$ (this is simply implemented by choosing a random number in the range from 0 to 1 and comparing this with the probability; if it is less, we make the move, otherwise we do not). The temperature T is then reduced according to the annealing schedule (which again must be determined by experimentation). The algorithm stops when some termination condition becomes

ISSN 1744-8050

satisfied (typically when no improvement has been made for a certain number of iterations or the maximum number of iterations has been reached).

4.2 Simulated Jumping (SJ)

A variation on Simulated Annealing is Simulated Jumping (Amin, 1999). It is based on fact that some materials containing both ferromagnetic and anti-ferromagnetic materials are known to have many metastable states. The theory is that for these types of materials, it is much harder to find a ground state (low energy state) just by cooling alone and, instead, a process of rapid heating and rapid cooling may be more likely to obtain such a low energy state. Thus, simulated jumping tries to exploit this concept in combinatorial optimisation problems. Rather than gradually decreasing (over a run) the probability of accepting an upwards move (as in simulated annealing), simulated jumping increases and decreases this probability many times over a run.

Pseudo code for Simulated Jumping is shown in Figure 2. The cooling and heating schedules are those suggested in Amin (1999) and may need to be adapted for different problems. The algorithm is the same as Simulated Annealing, except that if no move is made, the temperature is increased and the temperature is only decreased after a set number of moves/temperature increases. Simulated Jumping has been applied to the quadratic assignment problem, the asymmetric travelling salesman problem and channel assignment in mobile radio networks.

ISSN 1744-8050

```
SimulatedJumping(T0, $\gamma$, R, MaxCycles)
{
      //Typical values for parameters from Amin(1999)
      //T0 = 0.001, $\gamma$ = [0.001,0.2], R = 0.15, MaxCycles = 300
      x* = x = GenerateInitialSolution()
      T = T0

      do
      {
         for i = 1 to MaxCycles
         {
            Pick a neighbour y from the neighbourhood N(x) at random
            $\Delta$f = f(y) - f(x)
            if ($\Delta$f < 0) x = y
            else
            {
                  r = random number in range [0,1]
                  if (r < $e^{-\Delta f/T}$)     x = y        //Accept the change
                  else               T = T+R/i    //Heat the system
            }
            T = $\gamma$ * T //Cool the system
            if (f(x) < f(x*)) x* = x
         }
      }
      while not termination condition
}
```

Figure 2: Pseudo code for Simulated Jumping

4.3 GSAT and Walksat

GSAT (Selman et al. 1992, Selman & Kautz 1993a,b) and Walksat (Selman et al. 1994, Selman et al. 1997) are algorithms for dealing specifically with the SAT problem (a version of Walksat was also adapted for solving weighted MAX-SAT problems in Jiang et al. 1995). Both GSAT and Walksat make use of randomness to help them escape from local minima and plateaus by flipping a variable involved in a clause at random (although the way this is done for each is slightly different).

Pseudo code for the basic GSAT algorithm is given in Figure 3. The algorithm starts with a random solution x and then makes MAX_FLIPS changes to x (by flipping one boolean variable in the solution x, at a time; with probability *1 - noise*, that variable which decreases the maximum number of unsatisfied clauses is flipped, or with probability *noise* a randomly picked variable that is involved in one or more unsatisfied clauses is flipped), unless of course a solution that satisfies all the clauses is found, in which case this is returned. If a solution has not been found after

MAX_FLIPS, the algorithm restarts from a new random point. This continues until a solution is found, or the maximum number of restarts (MAX_TRIES) has been made.

```
GSAT(noise,MAX_FLIPS,MAX_TRIES)
{
      for i = 0 to MAX_TRIES-1
      {
            x = random assignment
            j = 0
            while (j < MAX_FLIPS) and (#unsat_clauses > 0)
            {
                  With probability(noise)
                  {
                      x = x with a variable flipped at random which
                            is involved in an unsatisfied clause
                  }
                  else
                  {
                      x = x with that variable flipped which leads to
                            the minimum number of unsatisfied clauses
                  }

                  j = j + 1
            }
            if (#unsat_clauses = 0) return x

      }
      return FALSE //Couldn't find an feasible assignment
}
```

Figure 3: Pseudo code for GSAT

Pseudo code for the basic Walksat is given in Figure 4. Walksat works in a similar fashion to GSAT, except that the way it chooses which variable to flip is slightly different. Walksat first chooses an unsatisfied clause at random. If no variable exists in the chosen clause such that it may be flipped with zero "breaks" (a "break" is defined to be a satisfied clause that becomes unsatisfied as a result of flipping a variable's value), then with probability *noise,* a variable in the chosen clause is picked at random and flipped (thus satisfying the chosen clause). Otherwise, the variable in the chosen clause which minimises the number of "breaks" is flipped. This continues until the maximum number of flips has been made (MAX_FLIPS) and then the search is restarted (MAX_TRIES) times or until a solution is found. The difference between Walksat and GSAT is that variables involved in many clauses are more likely to be flipped with Walksat, whereas GSAT considers all variables involved in unsatisfied clauses equally. It should be noted that we have only presented the most commonly known versions of GSAT and Walksat and that many other variants exist. The interested reader should refer to (Selman & Kautz 1993b, Selman et al., 1994, Gent & Walsh, 1993, Ginsberg & McAllester, 1994, Kask & Dechter, 1995, Cha & Iwama 1995, Frank 1996,1997, Frank et al. 1997, Jiang et al. 1995, Wei et al. 2004) for information on many other extensions of the basic versions of these algorithms (although this is by no means a complete list).

```
Walksat(noise,MAX_FLIPS,MAX_TRIES)
{
      for i = 0 to MAX_FLIPS-1
      {
            x = random initial assignment
            j = 0

            while (j < MAX_FLIPS) and (#unsat_clauses > 0)
            {
                  c = pick an unsatisfied clause at random

                  with probability(noise) and only if no variable may
                        be flipped with 0 breaks resulting (see text)
                  {
                      x = x with a variable in c chosen at random
                            flipped
                  }
                  else
                  {
                      x = x with that variable in c which minimises
                            the number of breaks flipped
                  }
                  j = j + 1
            }

            if (#unsat_clauses = 0) return x

      }
      return FALSE //Couldn't find an feasible assignment
}
```

Figure 4: Pseudo code for walksat

4.4 Greedy Randomized Adaptive Search Procedures (GRASP)

GRASP (Feo & Resende 1995) is a heuristic framework for local search. It now incorporates various meta-heuristics (see Resende and Ribeirom, 2003b). The main heuristic which separates this from other work, is a partly greedy, partly random start point construction heuristic (hence the name of the procedure), which can be used to start the search in a favourable region of the search space.

ISSN 1744-8050

```
GRASP()
{
      do
      {
            x = ConstructGreedyRandomizedSolution()
            x = LocalSearch(x)
            if (f(x) < f(x*)) x* = x
      }
      while not termination criteria
      return x
}

ConstructGreedyRandomizedSolution()
{
      x = {}
      while not complete solution and not termination criteria
      {
            RCL = MakeRestrictedCandidateList()
            <x_i,v> = SelectOneOfBestElementsAtRandom(RCL)
            x = x ∪ {<x_i,v>}
      }
      return x
}
```

Figure 5: Pseudo code for a basic GRASP (Feo & Resende, 1995)

Pseudo code for a simple GRASP and it's construction procedure is shown in Figure 5. The basic algorithm iteratively generates good start points (using the ConstructGreedyRandomizedSolution sub-procedure) and then improves the solution generated using a local search algorithm. If the best solution so far is improved, this is recorded. The initial solution is generated by constructing a partial solution, selecting at random between good candidate assignments for extending the partial solution. In this way, lots of randomly varying "good" start points may be generated. GRASP has been extended with many different meta-heuristics (for example, Path Relinking in Resende and Ribeiro, 2003a).

4.5 Iterated Local Search (ILS)

The simplest meta-heuristic for local search is to restart the algorithm from a new random start point. However, this means that all previous information gathered in the search is lost. A more sophisticated version of this approach, which utilises information collected in the previous runs of the local search algorithm, is the Iterated

ISSN 1744-8050

Local Search meta-heuristic (Stützle, 1999). The main motivation behind this approach is to utilise the previous local minima (in fact, in almost all implementations, the best found solution so far is used) and modify them (usually just making a set number of random moves from such a solution) to create new start points in order to increase the amount of time exploring more promising regions of the search space.

These mutations of previous local optima are commonly known as kick-moves, and they simply provide a way to escape from these local optima. The difference between this and simple random restarting is that previously found local optima (in most implementations, the best-found solution so far) are used to generate the new start point, with a few random modifications.

```
IteratedLocalSearch
{
      x = GenerateInitialSolution()
      x = LocalSearch(x)
      do
      {
            y = Modify(x,history)
            y = LocalSearch(y)
            x = AcceptanceCriterion(x,y,history)
      }
      while (termination condition not met)
}
```

Figure 6: Pseudo code for Iterated Local Search

Pseudo code for ILS is given in Figure 6. First, an initial solution (usually randomly generated) is generated by the GenerateInitialSolution function. The Local Search procedure is then used to improve upon this solution. Next, the Modify function takes the solution x, and changes it in some way (possibly based partly on the search history) and returns this new solution. The new solution is improved by the local search until a local minimum y is found and returned. Finally, this new solution y is then compared with the solution x, possibly taking into account information from the search history to decide whether to replace the old solution x with this new solution y and attempt to improve it further. If the Acceptance Criterion procedure accepts the new solution y, it returns y, otherwise it returns x. This process then continues until the termination condition is met.

The main limitation of this approach is that if the local optimum or previous best found solutions are not located close (in terms of the minimum number of moves between the two solutions) to the global optimum, then this method is probably no better or may be even worse than simple random restarting.

Many schemes can be made to fit into the ILS framework, by changing the Modify, LocalSearch and AcceptanceCriterion functions appropriately. However, here we just list the basic ILS algorithm. The main variations are in the way in which Modify changes the best found solution so far and how large the "kick-move" is (that is, how many random moves it comprises).

5 Population based algorithms

In this section, we describe meta-heuristics that store a population of solutions to try to improve the search ability of algorithms.

5.1 Ant Colony Optimization Algorithms (ACOs)

Ant algorithms (Dorigo et al. 1996, Gambardella et al. 1999, Stützle 1997, Taillard & Gambardella 1997) are based on the idea of having a population of solutions, with each solution worked on by an individual "ant", and with all the ants sharing a common data structure containing "pheromone information" accumulated over the course of the search. For example, in the Travelling Salesman Problem, Dorigo et al. (1996) place the pheromone information in a matrix, representing the amount of pheromone on each edge joining two cities. The higher the value for the pheromone trail on an edge, the more desirable the edge is.

Pseudo code for a simplified hybrid ant algorithm is given in Figure 7. The algorithm starts by giving each "ant" a random solution and then improving it using a local search algorithm. The pheromone trails are initialised to a value based on the cost of the best solution found so far. Then each ant performs a number of steps attempting to improve the current ant's solution - partly greedily according to the pheromone information, and partly randomly. The resulting solution from the modifications is then improved using local search. If the algorithm is in an intensification phase (this part of the algorithm is not shown in the pseudo code to simplify the pseudo code), then the best solution found during the modification steps and after the local search is set as the current ant's current solution. Otherwise the most recent solution is set as the current solution of each ant. This is repeated for all the "ants". If no improvement is made by any of the ants to their solutions, then the intensification is switched off (as the current area of the search space is not very promising). If the best solution so far has been improved, then intensification is turned on (as the current area of the search space is promising). All elements of the pheromone trail now have their values reduced (to simulate evaporation of a real pheromone trail). Next, those elements of the pheromone trail that are present in the best found solution so far have their values increased (to reinforce these good features of solutions). If, after a number of iterations, no improvement has been made to the best found solution so far then the algorithm "diversifies" (this part is also not shown in the pseudo code to aid simplicity) by reinitialising the pheromone trail data structure, and setting all but one of the ants' solutions to a new random start point, with the remaining ant having its solution set to the best found solution so far. This process continues until some termination condition is satisfied. It should be noted that this is only one example of an ant algorithm, and it should be stressed that there are many other variations in the literature.

```
AntAlgorithm(R,q,)
{
      //Foreach ant generate a starting solution
      foreach ant i do
      {
            $x_i$ = GenerateInitialStartPoint()
            $x_i$ = LocalSearch($x_i$)
      }

      //Intialise pheromone trails
      T = InitialisePheromoneTrails()

      do
      {
            foreach ant i do
            {
                  for r = 1 to R do $x_i$ = ApplyAntAlgorithmMove($x_i$,T)

                  //Improve the solution
                  $x_i$ = LocalSearch($x_i$)
            }

            UpdatePheromoneTrails(T,x)
      }
      while (not termination condition)
}

UpdatePheromeTrails(T,x*)
{
      //Simulate evaporation of the pheromone trails
      foreach (i,j) such that 1<=i,j<=N do $T_{ij} = (1 - \alpha_1)T_{ij}$

      //Reinforce pheromone trails using best solution found so far
      for i = 1 to N do $T_{ix^*[i]} = T_{ix^*[i]} + \alpha_2 / f(x_i^*)$

      return T
}

//Ant i, pheromone trail T are parameters
ApplyAntAlgorithmMove(i,T)
{
      withprobability (q)
      {  //perform exploitation
         choose a neighbouring solution $x_i^k$ from N($x_i$)
         partly randomly, such that the amount
         of pheromone in T is maximised
      }
      else
      {  //perform exploration
         choose a neighbouring solution $x_i^k$ from N($x_i$)
         partly randomly and partly randomly weighted
         towards those solutions with high amounts of
         pheromone in T
      }
      return x
}
```

Figure 7: Pseudo code for a simplified hybrid ant algorithm based on HAS-QAP (Gambardella et al. 1997)

ISSN 1744-8050

5.2 Scatter Search (SS)

Scatter Search (Cung et al. 1997) is a simple evolutionary heuristic which is very similar to memetic and genetic algorithms. Pseudo code for Scatter Search is given in Figure 8. To use Scatter Search, the *R* and *Q* parameters need to be set to appropriate values and a function for combining several solutions into one new solution (GenerateNewSolFromSols) in a random way needs to be defined. The idea is to keep a pool of *"elite" solutions*, and combine the *R* best of these *Q* elite solutions, then apply an "improvement operator" (typically some form of local search) to each one to generate a new solution. Then, if this solution is better than the worst of the elite solutions, it is inserted among the *Q* elite solutions, replacing the current worst elite solution, and the process continues until some stopping criterion is met.

```
ScatterSearch(R,Q)
{
      population = GenerateInitialPopulationOfQSolutions()
      while not termination condition
      {
            sols = SelectRBestSolsForCombining(population)
            x = GenerateNewSolFrom(sols)
            x = LocalSearch(x)
            population = InsertSolIntoPopulation(population,x)
      }
}
```

Figure 8: Pseudo code for Scatter Search

5.3 Path Relinking (PR)

A technique related to Scatter Search (it can easily be used as the GenerateNewSolFrom(sols) sub-procedure in Scatter Search) is path relinking (Glover & Laguna, 1997, Glover et al. 2000). The basic idea in path relinking is to attempt to find good solutions between two already good solutions, by exploring the path between the two good solutions. This exploits the fact that in many problems, good solutions are clustered near each other (in “a big valley” landscape (Boese et al. 1994) or share similar structure as in the proximate optimality principle of Glover & Laguna, 1997). This simple, relatively new idea can easily be used as a method for

ISSN 1744-8050

recombination of solutions in Scatter Search, and has also been used to improve the Genetic Algorithm of Reeves & Yamada (1998), as well as in Resende's greedy randomized adaptive search procedure (GRASP) in Resende & Ribeiro (2003). Pseudo code for a basic Path Relinking procedure is shown in Figure 9. Hamming distance which is the number of differences in the values assigned to the same variable in two solutions, can also be substituted for other methods for measuring the distance between solutions.

```
PathRelinking(x1,x2)
{
      x = x1
      x* = x1 if f(x1) < f(x2), else x2
      while xa • xb and not termination condition do
      {
            x = x' from N(x) such that f(x') is minimized and
                hamming distance(x',x2) < hamming_distance(x,x2)
            if f(x) < f(x*) x* = x
      }
      return x*
}
```

Figure 9: Pseudo code for a basic Path Relinking algorithm

5.4 Genetic Algorithms (GAs)

Genetic Algorithms (Holland, 1975) are the most famous population based method and have been applied to a large number of different types of problems. The idea stems from attempting to copy the way in which nature has evolved and selected the fittest individuals for reproduction, whilst occasionally mutating the chromosomes / genes of these individuals. To use a Genetic Algorithm (GA) to solve a problem, a cost function must be defined for evaluating potential solutions, together with a suitable representation for those solutions, and with crossover and mutation operators which must manipulate solutions in the chosen representation. The crossover operator must take two (or possibly more, but most GAs use only two) parents from the population and recombine them in some way, which is usually partly random, into a new valid solution. The mutation operator must take an existing solution from the population and make a small (possibly random) modification to it, also producing a new valid solution. As one might guess, how the designer of a GA represents solutions as chromosomes and how the crossover and mutation operators work are critical in how well the GA will work.

Whilst Genetic Algorithms have been used with some success on many problems, we believe that the most successful use of such algorithms is when a local search or similar heuristic is used in a hybrid scheme to help improve solutions produced by the GA. An example of such an approach is the Genetic Hybrid algorithm of Fleurant & Ferland (1994), where a Robust Tabu Search algorithm (see the later section on Tabu Search) is run for a set number of iterations to improve solutions generated by the Genetic Algorithm before they are inserted into the population (this algorithm has been successfully applied to the Quadratic Assignment Problem, finding some new best known solutions). The Genetic Hybrid algorithm is similar to the approach taken by a new group of algorithms called Memetic Algorithms, which we discuss in the next subsection.

Pseudo code for a basic Hybrid Genetic Algorithm (combined with Local Search) is given in Figure 10. The algorithm starts by creating an initial population of solutions, and then creating a new generation, by means of probabilistically selecting parents and individuals (this may be my means of a kind of roulette wheel mechanism which biases the selection towards fitter individuals) to perform crossover, mutation and reproduction a number of times until the new population has reached the predefined population size. Each solution generated is improved by applying a local search (or some similar heuristic) before re-insertion into the population. This process then continues until some termination condition is reached (e.g. a sufficiently good

ISSN 1744-8050

solution has been found, the maximum number of iterations has been reached or no improvement has been made for a number of iterations).

```
GeneticHybridAlgorithm(Pr,Pc,Pm)
{
      population = GenerateInitialPopulation()
      foreach solution in population do
            solution = LocalSearch(solution)
      do
      {
            next_population = {}
            for i=1 to population_size
            {
                  //Execute one of
                  with probability(Pr) //reproduce
                  {
                     solution = roulettewheelselection(population)
                  }
                  with probability(Pc) //crossover
                  {
                     sol1 = roulettewheelselection(population)
                     sol2 = roulettewheelselection(population)
                     solution = crossover(sol1,sol2)
                  }
                  with probability(Pm) //mutation
                  {
                     sol1 = roulettewheelselection(population)
                     solution = mutate(sol1)
                  }
                  improved = LocalSearch(solution)
                  next_population = next_population ∪ { improved }
            }
            population = next_population
      }
      while (not termination condition)
}
```

Figure 10: Psuedo code for a basic Hybrid Genetic Algorithm

5.5 Memetic algorithms (MAs)

Memetic algorithms (Moscato, 1989, 1993) combine ideas from genetic algorithms with more "aggressive" local search algorithms. The difference between GAs and MAs is that MAs are a more general concept than GAs, since memetic algorithms supposedly mimic "cultural evolution" rather than "genetic evolution" and therefore are not confined to the Genetic Algorithm framework. They may also incorporate many other types of algorithms and heuristics.

```
MemeticAlgorithm()
{
      population = GenerateInitialPopulation()
      foreach x in population
            x = LocalSearch(x)
      do
      {
            for i = 1 to #recombinations
            {
                  select two parents p1, p2 randomly from population
                  x = Recombine(p1,p2)
                  x = LocalSearch(x)
                  population = AddToPopulation(x)
            }
            population = Select(population)

            if Converged(population)
            {
                  foreach x in population \ { best } do
                        x = LocalSearch(Mutate(x))
            }
      }
      while (not termination condition)
}
```

Figure 11: Pseudo code for an example of a simple Memetic Algorithm (MA) (adapted from Merz and Freisleben, 1999)

Figure 11 shows pseudo code for a simple example of a memetic algorithm (this version has been successfully applied to the Quadratic Assignment Problem). The algorithm starts by generating a pool of random start points. The local search algorithm is then applied to each start point to improve it. Then two parents are selected randomly (without fitness bias) from this pool and combined using a recombination operator. After this, a local search algorithm is again applied to the resulting solution, which is then added to the population. This is repeated for the desired number of recombinations. Then the P best solutions are selected from the population and kept, throwing away any worse solutions. If the population has not changed for a constant number of iterations (typically around 30) or the average Hamming distance between solutions in the population drops below 10, then the population is deemed to have converged. When this happens, a mutation operator followed by the local search algorithm is applied to each solution in the population to restart / diversify the search, and the search continues as before until some termination condition is satisfied.

A very similar approach to memetic algorithms is the Genetic Hybrid algorithm of Fleurent & Ferland (1994), already covered in the previous section on Genetic Algorithms. These methods are also very similar to the elite solutions restarting from

ISSN 1744-8050

tabu search (Glover & Laguna, 1997), where a list of elite solutions is kept for generating new start points, possibly with additional information about the frequency of occurrence of solution attributes in good quality solutions. Many other similar hybrid approaches also exist, for example combining Simulated Annealing with Tabu Search (by representing the tabu list as penalties) and Genetic algorithms, as in Fox (1993).

5.6 Estimation of Distribution Algorithms (EDAs)

Estimation of Distribution Algorithms (Mühlenbein & Paass, 1996, Larranga & Lozano, 2002) are based on the idea of extracting global statistical information from selected solutions (often known as parents) and building a posterior distribution model of promising solutions based on the extracted information. New solutions are sampled from the distribution model and are then used to form the new population. Many different varieties of EDA algorithms have been developed for optimisation problems (see Larranga and Lozano, 2002). A variation of EDAs are Cross-entropy methods (Rubinstein & Kroese, 2004) where the probability models are built using techniques for estimating the probabilities of rare events.

```
EsimationOfDistributionAlgorithm()
{
      population = GenerateInitialPopulationRandomly()
      forall solution in population do
            solution = LocalSearch(solution)
      do
      {
            parents = SelectParents(population)
            prob_model = BuildProbabilityModel(parents)
            children = SampleSolutionsFromDistribution(prob_model)
            forall solution in children do
                  solution = LocalSearch(solution)
            population = Select(population ∪ children)
      }
      while (not termination condition)
}
```

Figure 12: Pseudo code for a Hybrid Estimation of Distribution Algorithm

In practice, EDAs are often combined with other heuristics such as local search, guided local search and genetic algorithms to solve hard optimisation problems (for example, see Zhang et al. 2004, Zhang et. al. 2003a,b). Figure 12 shows high level pseudo code for a Hybrid Estimation of Distribution Algorithm combined with Local Search.

6 Neighbourhood based algorithms

In this section, we describe meta-heuristics which help prevent local search algorithms becoming trapped in local minima (and also some that speed up the search process) by restricting the neighbourhood or expanding the local search neighbourhood when they become trapped.

6.1 Variable Neighbourhood Search (VNS)

Variable Neighbourhood Search (Mladenovic & Hansen, 1997) has several local search neighbourhoods of increasing size available to it. It begins at some initial start point (usually randomly chosen). It then picks a neighbouring solution at random from the smallest-sized neighbourhood and applies a local search until a local minimum is obtained. Then, when the solution has not been improved, the next largest neighbourhood is utilised in the same way (the neighbourhood of the local search procedure is the same however). If an improving solution is obtained, then the smallest neighbourhood is again utilised. Otherwise the next largest neighbourhood is tried, until the maximum-sized neighbourhood has been reached. Pseudo code for VNS is given in Figure 13.

```
VariableNeighbourhoodSearch($N_1$(),$N_2$(),..,$N_{kmax}$())
{
      x = GenerateInitialSolution()
      k = 1
      do
      {
            y = random solution picked from $N_k$(x)
            z = LocalSearch(y)
            if (f(z) < f(x))
            {
                  x = z
                  k = 1 //improved solution => use smallest N(x)
            }
            else if (k < kmax) //no improved solution found
```

ISSN 1744-8050

```
            {
                  k = k + 1 //=>use next largest neighbourhood
            }
      }
      while (not termination condition)
}
```

Figure 13: Pseudo code for basic Variable Neighbourhood Search (VNS)

6.2 Tabu Search (TS)

```
BasicTabuSearchWithBestImprovedAspirationCriterion()
{
      x = GenerateInitialSolution()
      x* = x;
      TabuList = {}

      while (not termination condition)
      {
         //note: 2nd condition is best-improved aspiration criterion
         Pick best y from N(x) such that (not Tabu(x,y,TabuList))
                                         or (f(y) < f(x*))
         if (f(y) < f(x*))
            x* = y

         x = y

         TabuList = TabuList ∪ {attribute of x or move from x to y}

         if (size of TabuList > MaxTabuListSize)
            remove oldest element from TabuList
      }

}

Tabu(x,y,TabuList)
{
      foreach element t in TabuList
            if (move from x to y is tabu because y contains t or the
                move itself is tabu as it reverses an earlier move)
                  return true
      return false
}
```

Figure 14: Pseudo code for basic Tabu Search with the best-improved aspiration criterion

Tabu Search (Glover,1989,1990, Glover et al. 1993, Glover & Laguna, 1997) is a framework for local search which incorporates many different ideas. The main idea is that of the tabu list, where a list of tabu attributes (such as arcs between cities in the TSP or variables flipped in the SAT problem) of previously visited solutions or moves used is maintained, so that the local search algorithm may escape from local minima, by disallowing moves to previous solutions that possess these "already used/explored" attributes. In most successful implementations, if there exists a move which is tabu but which never the less improves the best found solution so far, then the tabu status of the move is ignored and the move is made to generate the new best found solution. This is an example of an aspiration criterion, and is known as the *"improved-best" aspiration criterion*. Pseudo code for a basic tabu search algorithm is shown in Figure 14.

Protagonists of Tabu Search would claim that any local search algorithm which uses some form of memory based on the previous history of the search to influence the future direction of the search is a member of the tabu search family. However, we believe that while there may be some element of truth in this, there is also the consideration that if one tries hard enough, it is always possible to draw parallels between different search methods. For this reason, we have only listed the basic elements of tabu search in this section, and some examples of successful tabu search algorithms in the next two subsections, although the interested reader may refer to Glover & Laguna (1997) for a detailed discussion of the many ideas in the area of tabu search.

6.2.1 Robust Tabu Search (RTS)

Robust Tabu Search (Taillard, 1991, 1997) is an enhanced version of the basic tabu search scheme, which uses a randomly varying length tabu list and a form of long term memory. The maximum tabu list length is varied by plus or minus some percentage (10% in the QAP) around some fixed value (the number of elements in a solution permutation in the QAP) every time a local search move is made. The long term memory forces solution attributes which have not been present for a certain number of moves (e.g. 4.5 n^2 for the QAP, where n = permutation solution size) back into solutions. This is done by making any move which does not introduce the desired attribute "tabu" and therefore disallowed (unless the best-improved aspiration criterion is applicable). Robust Tabu Search has been shown to be successful in tackling the QAP. Recently, Smyth et al. (2003) have applied Iterated Robust Tabu Search (with Iterated Local Search added to Robust Tabu Search) to gain good results on the MAX-SAT problem.

6.2.2 Reactive Tabu Search (ReTS)

Reactive Tabu Search is yet another notable enhancement of the basic tabu search scheme. This scheme is quite complicated, so it is not possible to give full details here. The interested reader should refer to Battiti & Techchiolli (1994, 1995). The main idea is that the tabu list length is increased, if there are many solutions being revisited, and shortened, when not so many solutions are revisited. In this way, the algorithm maintains a list length which is best suited to the current problem and the area of the search space. The second feature of Reactive Tabu Search is that it makes a sequence of random moves if the algorithm finds that it is trapped in an area of the search space (this is again determined by counting the number of times solutions are revisited) which for some reason cannot be escaped from just by using a simple tabu search strategy. This part of ReTS resembles the idea of Iterated Local Search, where a similar method is employed to escape from local minima, rather than using a random restart from a completely new solution.

6.3 Fast Local Search (FLS) , "don't look bits" and Elite Candidate Lists

Fast Local Search (Voudouris,1997) is an generalisation/adaptation of an earlier scheme known as "don't look bits" (Bentley, 1992) which is designed to be used to speed up "first improvement" local search algorithms. Together these two similar heuristic speed-ups have been successfully applied to the TSP, partial constraint satisfaction problems (Radio Link Frequency Assignment Problem) and recently, the QAP with Iterated Local Search (Stützle,1999).

```
FastLocalSearch(x)
{
      foreach m in movesofN(x) don't_look(m) = false
      UnscannedNs = movesofN(x)

      do
      {
            m = pick an element m of movesofN(x) at random,
                such that don't_look(m) = false
            UnscannnedNs = UnscannnedNS - {m}

            if (delta(m(x)) <= 0)
            {
                //Don't bother checking inverse of move
                don't_look(inverse(m)) = true
                x = m(x) //Execute move m(x)
                forall moves m' such that m' affected by m
                {
                   don't_look(m') = false //Re-activate those moves
                }
            }
            else
            {
                //Move is currently poor, so don't check it next time
                don't_look(m(x)) = true
            }
      }
      while (there exists a move m(x) in UnScannedNs,
             such that don't_look(m) = false)

      return x
}
```

Figure 15: Pseudo code for basic Fast Local Search / "don't look bits" procedure

The idea of FLS and "don't look bits" is to speed up neighbourhood search by ignoring parts of the neighbourhood which are unlikely to yield better solutions (based on previous evaluations of the neighbourhood). This is implemented by simply storing a "don't look bit" with each element or sub-component of the neighbourhood. If during scanning of the neighbourhood, an element of the neighbourhood yields an upwards move, the bit is turned on, and that element of the neighbourhood is no longer evaluated until the bit is turned off again. The bit is only turned off again when some event occurs which makes it likely that the move may now have become desirable again, e.g. a move is made which affects that element of the neighbourhood in some way or a penalty is imposed. When this occurs, the "don't look bit" is flipped back to zero (off), and evaluation of this element of the neighbourhood is no longer ignored (at least, until the don't look bit is again turned on). Figure 15 shows pseudo code giving a basic idea of how such a scheme should work in general.

FLS and "don't look bits" are also similar to the elite candidate list strategy used in tabu search (Glover et al. 1993, Glover & Laguna, 1997). In this method, a list of "elite moves" is constructed by examining the whole or part of the neighbourhood and this list of moves is used until the moves become too poor in quality, when a new list of elite moves is built and the process is then reiterated throughout the search.

All three of these techniques take advantage of the fact that, in many applications, a move's status, in terms of whether it is a good or a bad quality move, may be highly likely to stay the same, even after several other moves have been made. In problems where this is not the case, then these techniques are obviously not likely to be useful, but in problems where the size of the neighbourhood is massive, these techniques may make a large saving in running time.

7 Weighted and Penalty based algorithms

In this section, we describe algorithms that use penalties or weights to modify the objective function that the local search is optimising in order to help them escape from local minima. The idea behind all the algorithms presented in this section is to try to “fill-in” a local optimum by modifying the weight or penalty terms in the objective function and thus increasing the cost associated with the local optimum.

7.1 GENET and other Weighted Constraint algorithms

GENET (Tsang & Wang, 1992, Davenport, 1997) is a heuristic repair method, which modifies a weighted objective function in order to escape from local minima. To use GENET to solve a particular problem, each constraint in the problem must have a weight associated with it, along with a violation function V, which defines the degree to which the constraint is violated (this may be as simple as 1 for violated or 0 for satisfied, but if the constraint may be violated, such that more than one variable will require its value to be changed, it is much more efficient if this function gradually decreases as the constraint becomes closer to being satisfied). GENET uses these violation functions as it attempts to maximise the (negatively) weighted sum of violation functions for all the constraints in the problem (this is referred to as the *energy* for historical reasons (GENET was originally a Neural Network)).

Pseudo code for the basic limited sideways GENET scheme is shown in Figure 16. The algorithm goes through one variable at a time, trying to maximise the energy of GENET by modifying the current label for the current variable it is examining. If more than two consecutive *sideways moves* (moves to solutions of equal cost) are made, the algorithm is regarded as being stuck in a local minimum and all the weights of violated constraints in that local minimum state are decreased by 1. The algorithm then continues in this manner until some termination condition is satisfied or all the constraints are satisfied.

For more information on GENET the interested reader may refer to Davenport (1997). For the history of GENET's development, the interested reader may refer to (Tsang & Wang, 1992, Tsang 1993, Davenport et al. 1994, Tsang et al. 1999). Much work has been carried out on extending GENET. This includes adding lazy constraint consistency to GENET (Stuckey & Tam, 1996, 1997) and introducing variable ordering strategies (Tam & Stuckey, 1998) to attempt to improve the performance, as well as various other schemes based on adding additional constraints and "nogood" constraints to the problem (Lee et al. 1995, 1996, 1998). For a study of GENET compared to Tabu Search on a group of partial constraint satisfaction problems, the

ISSN 1744-8050

interested reader should refer to Boyce et al. (1995). GENET has also been shown to belong to the class of Discrete Langrangian Search algorithms by Choi et al. (1998).

```
GENET(Z,D,C,V)
//Z = variables, D = domains of variables,
//C = set of constraints
//V = set of violation functions, 1 per constraint
{
      foreach $x_i$ in Z, $x_i$ = a random element from $D_{x_i}$
      foreach $c_i$ in C, $w_{c_i}$ = -1
      sideways = 0

      while  #{c in C | c is not satisfied} > 0 and
             sideways < 2 and
             not termination condition
      {
            foreach $x_i$ in Z
            {
               $x_i$ = value from $D_{x_i}$ such that it maximises sum of
                    $V_{c_i}(x_0..x_n)*w_{c_i}$ of violated constraints on $x_i$
                    (Break ties randomly)

               if (sum($V_{c_i}(x_0..x_n)*w_{c_i}$) of violated constraints stays
                   the same and value of $x_i$ is different from before)
                   sideways = sideways + 1
               else if (value of $x_i$ is different from before)
                   sideways = 0
            }

            foreach violated constraint $c_i$ in C
                  $w_{c_i}$ = $w_{c_i}$ - 1
      }
      if (#{$c_i$ in C | $c_i$ is not satisfied} = 0)
            return true       //solution found
      else
            return false      //no solution found
}
```

Figure 16: Pseudo code for GENET, an example of a weighted constraint solver

As well as GENET, many other algorithms based on the same principle have been used for solving problems with simpler types of constraints than those used by the GENET researchers. These include Breakout (Morris, 1993), where a weight on each clause (nogood) is increased every time a local minimum solution is reached; otherwise, a move is made to reduce the cost function (sum of the weights of violated constraints). However, the really important point made by Morris (1993) is that, if every time a local minimum is found, the weight of a nogood representing the complete current solution is increased, then this algorithm can be shown to be complete (although it should be noted that this result does not extend to weighted and penalty based algorithms, where only part of the current solution's cost is increased (e.g. the weight of a nogood tuple of a violated constraint)).

Another algorithm to use weighted constraints is an extension of the GSAT algorithm with weights (Selman & Kautz, 1993a,b). In this algorithm, the weights of all clauses not satisfied at the end of a "try" (a run of algorithm beginning at a (usually random) start point) are increased. Later work, involving weights on each clause for GSAT, increased the weights of all unsatisfied clauses after each flip (Frank, 1996). Later, this scheme was extended to also decrease all weights of clauses after each flip as well (Frank, 1997), although we believe that this "short term" weighted clause approach is probably not practical owing to excessive CPU requirements of decreasing *every* clause's weight after *every flip* of a variable.

7.2 Guided Local Search (GLS)

Guided Local Search (see Voudouris (1997) for a more detailed description) is a penalty based meta-heuristic that sits on top of a local search algorithm to help it escape from local minima and plateaus. When the given local search algorithm settles in a local optimum, GLS modifies the objective function using a scheme that will be explained below. Then the local search will operate using an augmented objective function, which is designed to bring the search out of the local optimum. The key is in the way that the objective function is modified.

7.2.1 Solution features

Solution features are defined to distinguish between solutions with different characteristics, so that poor characteristics can be penalised by GLS, and hopefully removed by the local search algorithm. The choice of solution features therefore depends on the type of problem, and also to a certain extent on the local search

ISSN 1744-8050

algorithm. We define for each feature f_i a cost function c_i (which often comes from the objective function). Each feature is also associated with a penalty p_i (initially set to 0) to record the number of occurrences of the feature in local minima. Examples of features are unsatisfied clauses in the SAT and weighted MAX-SAT problems, and location-facility assignments in the QAP. At the implementation level, we define for each feature i an Indicator Function I_i indicating whether the feature is present in the current solution or not:

$$I_i(x) = \begin{cases} 1, & \text{solution } x \text{ has property i} \\ 0, & \text{otherwise} \end{cases} \qquad \textbf{(1)}$$

Concrete examples of indicator functions for the SAT, the weighted MAX-SAT and Quadratic Assignment Problems are given in (2) and (3), respectively.

$$I_{C_i}(x) = \begin{cases} 1, & \text{if solution } x \text{ does not satisfy clause } C_i \\ 0, & \text{otherwise} \end{cases} \qquad \textbf{(2)}$$

$$I_{<x_i,k>}(x) = \begin{cases} 1, & \text{if location } x_i \text{ contains facility } k \text{ (the } i\text{th element of the permutation } x \text{ is } k) \\ 0, & \text{otherwise} \end{cases} \qquad \textbf{(3)}$$

Examples of cost functions for features for the SAT & weighted MAX-SAT and QAP are given in (4) and (5), respectively.

$$Cost_{C_i}(x) = \begin{cases} Wc_i & \text{for weighted MAX-SAT problems} \\ 1, & \text{for SAT problems} \end{cases} \qquad \textbf{(4)}$$

$$Cost_{<x_i,k>}(x) = \sum_{j=0}^{n} a_{ij} b_{kx_j} \text{ for the feature } < x_i, k > \text{ representing when location } x_i \text{ contains facility } k \qquad \textbf{(5)}$$

7.2.2 Selective penalty modifications

When the Local Search algorithm returns a local minimum x, GLS penalises all those features (through increments to the penalty of the features) present in that solution which have maximum utility, *util(x,i)*, as defined in (6). See Figure 17 for pseudo code of the overall GLS algorithm.

$$util(x,i) = I_i(x)\frac{c_i(x)}{1+p_i} \tag{6}$$

The idea is to penalise features that have high costs, although the utility of doing so decreases as the feature is penalised more and more often.

```
Guided_Local_Search (x,f,a,λ,N)
{
          for all p_i, p_i = 0
          x* = x = random assignment or permutation (QAP)
          do
          {
           g = f augmented as in (7)
           x = Local_Search(x,f,g,N)
           Features_To_Penalise = {i|util(x,i) is maximised &
                                   I_i(x) = true }
           for each j in Features_To_Penalise
           {
            p_j = p_j + 1
           }
          }
          while (not termination condition)
          return x*
}

Local_Search(x,f,g,N)
{
          do
          {
           y = solution in N(x) such that h(x) is minimised,
                breaking ties randomly
           Δg = g(y) - g(x)
           if (Δg <= 0) x = y
           if (Δg = 0) sideways = sideways + 1
           else        sideways = 0
           if (f(x) < f(x*)) x* = x
          }
          while (Δg <= 0) and (sideways < 2)

          return x
}
```

Figure 17: Pseudo code for Guided Local Search

GLS uses an augmented cost function (7), to allow it to guide the Local Search algorithm out of the local minimum, by penalising features present in that local minimum. The idea is to make the local minimum more costly than the surrounding search space, where these features are not present.

$$g(x) = f(x) + \lambda \cdot a \cdot \sum_{i=1}^{m} I_i(x) \cdot p_i \tag{7}$$

The parameter λ may be used to alter the intensification of the search for solutions. A higher value for λ will result in a more diverse search, where plateaus and basins are searched more coarsely; a low value will result in a more intensive search for the solution, where the plateaus and basins in the search landscape are searched in finer detail. The coefficient *a* is used to make the penalty part of the objective function balanced relative to changes in the objective function and is problem specific. A simple heuristic for setting *a* is simply to record the average change in objective function up until the first local minimum, and then set *a* to this value divided by the number of GLS features in the problem instance.

Recently Mills (2002) has described an Extended Guided Local Search (EGLS) which utilises random moves and an aspiraton criterion designed specifically for penalty based schemes. The resulting algorithm improved the robustness of GLS over a range of parameter settings, particularly in the case of the QAP (see Mills et al. 2002). A general version of the GLS algorithm, using a min conflicts based hill climber (Minton et al. 1992) and based partly on GENET (Davenport, 1997) for constraint satisfaction and optimisation, has also been implemented in the Computer Aided Constraint Programming project (see Tsang et al. 1999 for an overview of this project).

7.3 Discrete Langrangian Multipliers (DLM)

The Discrete Langrangian Multiplier search algorithm is based on a modified mathematical theory from continuous optimisation. DLM associates a "Langrangian multiplier" with each constraint in the problem. This is increased each time DLM reaches a local minimum. As one can easily see, this is almost exactly what GENET and other weighted constraint algorithms do, and in fact GENET has been shown to belong to the same class of Langrangian-based search algorithms by Choi et al. (1998).

DLM has been applied to a number of problems, including the weighted MAX-SAT problem (Wah & Shang, 1997), and the SAT problem (Shang & Wah, 1998), as well as others, such as some continuous and mixed integer programming problems (Shang, 1997). The theory is that the algorithm is maximising the Langrangian multipliers (performing ascent in the Langrangian multiplier space), while minimising the objective function of the problem it is trying to solve (performing descent in the space of feasible solutions).

To gain good performance from DLM, it has been shown to be important periodically to reduce the Langrangian multipliers (Shang & Wah, 1998). An ad hoc "trap" escaping strategy has been added to DLM to improve its performance (Wu & Wah, 1999) on hard SAT benchmark problems. This strategy increases the Langrangian multipliers more than usual for clauses, that become unsatisfied more frequently. A slightly more general scheme, which performs the same job as the "trap escaping" strategy, is given in (Wuh & Wah, 2000), where a queue of previously visited solutions is maintained and then the number of previously visited solutions which are within a certain Hamming distance (one, in that paper) is added as a penalty to the objective function. For details of this and other extensions to DLM and many applications to other problems and the theory behind it, the interested reader should refer to (Shang, 1997, Wu, 1998, Wu, 2001).

7.4 Tabu Search With Penalties

Tabu Search (see Section 6.2), has also been suggested as a possible penalty based algorithm, by associating a penalty in the objective function with each item in the tabu list (Fox, 1993). In many ways, this is a very similar approach to Guided Local

Search, except that Guided Local Search penalises only a subset of those features found in local minima, whereas a tabu penalty algorithm would penalise all items in the tabu list.

Another technique from the tabu search community called *Frequency based memory* (Glover & Laguna, 1997) uses penalties added to the objective function to penalise solution attributes or moves, if they occur more frequently (the more often such an attribute occurs in a solution or such a move is made, the higher the penalty).

8 Conclusion

In this paper, we have attempted to survey practical meta-heuristics which may be used to escape or avoid local optima in local search algorithms. Most of the meta-heuristics in this paper work because they exploit the fact that good local optima solutions tend to be clustered near the global optimum (for example, see Boese et al. (1994), or later Reeves (1999)). This is called the proximate optimality principle (POP) by Glover and Laguna (1997). All these meta-heuristics exploit this property by using one or more of the following mechanisms:

- Making ***random non-improving moves*** (for example in GSAT, Selman et al. 1992) to allow solutions close to the current solution to be visited even if the current solution is in a local optima or plateau.
- ***Augmenting the objective function***, so that the current local optimum is no longer a local optimum with respect to the new objective function (for example, GENET in Davenport et al. 1994, Guided Local Search in Voudouris, 1997)
- Maintaining some kind of ***memory*** such as a ***tabu list*** of attributes of already visited solutions and making it ***tabu*** to visit such solutions (for example, in

Tabu Search, Glover 1989,1990), so that previously visited solutions are not revisited.

- Maintaining a ***population of good solutions*** and then recombining two or more of them (possibility using a ***probability model***, for example, Estimation of Distribution Algorithms, Larranaga and Lozano, 2002) in order to produce a new start point for local search (for example, Genetic Algorithms, Holland, 1992, or more recently Path Relinking, which explores the trajectory between two or more solutions).

Many challenges remain in meta-heuristic research, but we believe the key challenges at present are now to take existing meta-heuristics and turn them into usable algorithms, suitable for solving real world industrial problems. To this end, some of the most desirable features in meta-heuristics are:

- ***Convergence to global optima***, given sufficient time and resources.
- ***General applicability***, so they can deal with any problem defined in a suitably rich language, such as EaCL (Mills et al.1998, 1999) or OPL (Van Hentenryck, 2002).
- ***Suitability for parallel processing***, i.e. suitable for running on large computational grids.

Acknowledgements

This research was partially funded by EPSRC grants GR/L20122 and GR/M46297 and the University of Essex Research Promotion Fund. We would also like to thank Jian Sun & Tim Gosling for useful discussions on EDA algorithms.

References

Amin, A. (1999). Simulated Jumping. In *Annals of Operations Research, 86*, pages 23-38.

Battiti, R. & Tecchiolli, G. (1994). The Reactive Tabu Search. In *ORSA Journal on Computing, 6(2)*, pages 126-140.

Battiti, R. & Tecchiolli, G. (1995). Local search with memory: Benchmarking RTS. In *Operations Research Spektrum, 17(2/3)*, pages 67-86.

Bent, R. and Van Hentenryck, P. (2004). A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows. To appear in *Transportation Science*.

Bentley, J.L. (1992). Fast Algorithms for Geometric Travelling Salesman Problems. *ORSA Journal on Computing, 4(4)*, pages 387-411.

Burkard, R.E., Karisch, S.E., & Rendl, F. (1997). QAPLIB - A Quadratic Assignment Problem Library. In *Journal of Global Optimization,* 10, pages 391-403.

Boese, K., Kahng, A.B., & Muddu, S. (1994). A New Adaptive Multi-Start Technique for Combinatorial Global Optimizations. In *Operations Research Letters*, 16(2), Sept 1994, pages 101-113.

Boyce, J.F., Dimitropoulos, C.H.D., vom Scheidt, G., & Taylor, J.G. (1995). GENET and Tabu Search for Combinatorial Optimization Problems. In *World Congress on Neural Networks*, Washington D.C.

Cha, B. & Iwama, K. (1995). Performance tests of local search algorithms using new types of random cnf formulas. In *Proceedings of IJCAI'95,* pages 304-310.

Choi, K.M.F., Lee, J.H.M. & Stuckey, P.J. (1998). A Lagrangian Reconstruction of a Class of Local Search Methods. In *Proceedings of 10th IEEE International*

Conference on Tools with Artificial Intelligence, IEEE Press, Taipei, Taiwan, ROC, pages 166-175.

Cung, V., Mautor, T., Michelon, P. & Tavares, A. (1997). A Scatter Search Based Approach for the Quadratic Assignment Problem. In *Proceedings of IEEE International Conference on Evolutionary Computation and Evolutionary Programming*, Indianapolis, U.S.A., pages 161-170.

Davenport, A.J., Tsang, E.P.K., Wang, C.J. and Zhu, K. (1994). GENET. A connectionist architecture for solving constraint satisfaction problems by iterative improvement. In *Proceedings of AAAI-94,* pages 325-330.

Davenport, A. (1997). *Extensions and Evaluation of GENET in Constraint Satisfaction*, Ph.D. thesis. Department of Computer Science, University of Essex, U.K.

Dorigo, M., Maniezzo, V. and Colorni, A. (1996). The Ant System: Optimization by a colony of cooperating agents. In *IEEE Transactions on Systems, Man, and Cybernetics - Part B, 26*, pages 1-13.

Feo, T.A. & Resende, M.G.C. (1995). Greedy Randomized Adaptive Search Procedures. In *Journal of Global Optimization*, vol. 6, pages 109-133.

Fleurent, C., Ferland, J.A. (1994). Genetic Hybrids for the Quadratic Assignment Problem. In Pardalos, P., Wolkowicz, H., (eds.), *Quadratic Assignment and Related Problems*, Vol. 16, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 173-187.

Fox, B.L. (1993). Integrating and accelerating tabu search, simulated annealing, and genetic algorithms. In *Annals of Operations Research, 41*, pages 47-67.

Frank, J. (1996). Weighting for Godot. Learning heuristics for GSAT. In *Proceedings of AAAI'96*, pages 338-343.

Frank, J. (1997). Learning Short-Term Weights for GSAT. In *Proceedings IJCAI '97,* pages 384-389.

Frank, J., Cheeseman, P. and Stutz, J. (1997). When Gravity Fails: Local Search Topology. In *Journal of Artificial Intelligence Research, 7*, pages 249-291.

Freuder, E.C. & Wallace, R.J. (1992). Partial constraint satisfaction, *Artificial Intelligence, 58, Nos. 1-3 (Special Volume on Constraint Based Reasoning)*, pages 21-70.

Gambardella, L.M., Taillard, E.D. & Dorigo, M. (1999). Ant Colonies for the QAP. Accepted for publication in the *Journal of the Operational Research Society, 50*, pages 167-176.

Gent, I.P., van Maaren, H. & Walsh, T. (2000). SAT2000, *Highlights of satisfiability research in the year 2000*, Frontiers in Artificial Intelligence and Applications, IOS Press.

Gent, I. & Walsh, T. (1993). Towards an Understanding of Hill-climbing Procedures for SAT. In *Proceedings of AAAI-93*, pages 28-33.

Ginsberg, M. & McAllester, D. (1994). GSAT and Dynamic Backtracking. In *Fourth Conference on Principles of Knowledge Representation and Reasoning (KR-94)*, pages 226-237.

Glover, F. (1989). Tabu search Part I. *Operations Research Society of America (ORSA), Journal on Computing, 1*, pages 109-206.

Glover, F. (1990). Tabu search Part II. *Operations Research Society of America (ORSA), Journal on Computing, 2*, pages 4-32.

Glover, F. & Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers.

Glover, F., Laguna, M. & Martí, R. (2000). Fundementals of Scatter Search and Path Relinking. In *Control and Cybernetics*, Volume 29, Number 3, pages 653-684.

Glover, F., Taillard, E. & de Werra, D. (1993). A user's guide to tabu search. In *Annals of Operations Research, 41*, pages 3-28.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, 1975. Also second edition, The MIT Press 1992.

Jiang Y., Kautz H., and Selman B. (1995). Solving Problems with Hard and Soft Constraints Using A Stochastic Algorithm for MAX-SAT, *1st International Joint Workshop on Artificial Intelligence and Operations Research*, 1995.

Kask, K & Dechter, R. (1995). GSAT and Local Consistency. In *Proceedings of IJCAI'95*, pages 616-622.

Kautz, H. & Selman, B. (1996). Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In *Proceedings of AAAI-96*, pages 1194-1201. AAAI Press, 1996.

Kilby, P., Prosser, P. & Shaw, P. (1997). Guided Local Search for the Vehicle Routing Problem. In *Proceedings of the 2nd International Conference on Metaheuristics*, pages 21-24.

Kirkpatrick, S., Gelatt, C.D. & Vecchi, M.P. (1983). Optimisation by simulated annealing. In *Science 220*, pages 671-680.

Koza, J.R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

Koza, J.R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press.

Larranaga, P. & Lozano, J.A (2002). *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, Kluwer Academic Publishers, Boston.

Lau, T.L. (1999). *Guided Genetic Algorithm*, PhD Thesis, Department of Computer Science, University of Essex, U.K.

Lee, J.H.M., Leung, H.F. & Won, H.W. (1995). Extending GENET for Non-Binary Constraint Satisfaction Problems. In *Seventh IEEE International Conference on Tools with Artificial Intelligence*, IEEE Computer Society Press, Washington D.C., USA, pages 338-343.

Lee, J.H.M., Leung, H.F. & Won, H.W. (1996). Towards a More Efficient Stochastic Constraint Solver. In *Second International Conference on Principles and Practice of Constraint Programming*, Springer-Verlag, LNCS 1118, Cambridge, Massachusetts, USA, pages 338-352.

Lee, J.H.M., Leung, H.F. & Won, H.W. (1998). Performance of a Comprehensive and Efficient Constraint Library using Local Search, *11th Australian Joint Conference on Artificial Intelligence*, Springer-Verlag, LNAI 1502, Brisbane, Australia, pages 191-202.

Lesaint, D., Voudouris, C., Azarmi, N., Alletson, I. and Laithwaite, B. (2003). Field workforce scheduling. In *BT Technology Journal, 21,* Kluwer Academic Publishers, pages 23-26.

McAllester, D., Selman, B. & Kautz, H. (1997). Evidence for Invariants in Local Search. In *Proceedings AAAI-97*, Providence, Rhode Island, pages 321-326.

Merz, P. and Freisleben, B. (1999). A Comparison of Memetic Algorithms, Tabu Search, and Ant Colonies for the Quadratic Assignment Problem. In *Proceedings of the 1999 International Congress of Evolutionary Computation (CEC'99)*, IEEE Press, pages 2063-2070.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, M. & Teller, E. (1956). Equation of steady-state calculation by fast computing machines. In *Journal of Chemical Physics, 21*, pages 1087-1092.

Minton, S., Johnson M.D., Philips A.B. and Laird P. (1992). Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. In *Artificial Intelligence, 58*, pages 161-205.

Mills, P. (2002). *Extensions to Guided Local Search*, PhD Thesis, Department of Computer Science, University of Essex, U.K.

Mills, P. & Tsang, E.P.K. (2000). Guided local search for solving SAT and weighted MAX-SAT problems. In *Journal of Automatic Reasoning, 24, Special Issue on Satisfiability Problems*, Kluwer, pages 205-223.

Mills, P. & Tsang, E.P.K. & Ford, J. (2003). Applying an Extended Guided Local Search on the Quadratic Assignment Problem. In *Annals of Operations Research, 118*, Kluwer Academic Publishers, pages 121-135.

Mills, P., Tsang, E.P.K., Williams, R., Ford, J. & Borrett, J. (1998). EaCL 1.0: an easy abstract constraint programming language, Technical Report CSM-321, University of Essex, Colchester, U.K.

Mills, P., Tsang, E.P.K., Williams, R., Ford, J. & Borrett, J. (1999). EaCL 1.5: An Easy Abstract Constraint Optimisation Programming Language, Technical Report CSM-324, University of Essex, Colchester, U.K.

Mladenovic, N. & Hansen, P. (1997). Variable Neighborhood Search. In *Computers in Operations Research, 24*, pages 1097-1100.

Morris, P. (1993). The breakout method for escaping from local minima. In *Proceedings of AAAI-93*, AAAI Press/The MIT Press, pages 40-45.

Moscato, P. (1993). *On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms*, C3P Report 826, Caltech Concurrent Computation Program, Caltech, California, U.S.A.

Moscato, P. (1993). An introduction to population approaches for optimization and hierarchical objective functions: A discussion on the role of tabu search. In *Annals of Operations Research, 41*, pages 85-121.

Mühlenbein, H. & Paass, G. (1996). From Recombination of Genes to the Estimation of Distribution Part 1, Binary Parameter. In *Proceedings of the 4th international conference on Parallel Problem Solving from Nature-PPSN IV*, H-M Voigt et al, Ed., Lecture Notes in Computer Science 1141, Springer-Verlag, pages 178-187.

O'Reilly, U.-M., Oppacher, F. (1994). Program Search with a Hierarchical Variable Length Representation: Genetic Programming, Simulated Annealing and Hill Climbing. In *Parallel Problem Solving from Nature III*, Davidor, Schwefel, Manner (Eds), Springer Verlag (LNCS), Berlin, pages 397-406.

Reeves, C.R. (1999). Landscapes, operators and heuristic search. In *Annals of Operations Research, 86*, pages 473-490.

Reeves, C.R. and Yamada, T. (1998). Genetic Algorithms, Path Relinking and the Flowshop Sequencing Problem. In *Evolutionary Computation Journal* 6, pages 45-60.

Resende, M.G.C. & Ribeiro, C.C. (2003a). GRASP and path-relinking: Recent advances and applications. AT & T Labs Research Technical Report, April 6, 2003.

Resende, M.G.C. & Ribeiro, C.C. (2003b). Greedy Randomized Adaptive Search Procedures. In *Handbook of Metaheuristics*, Kluwer Academic Publishers, pages 219-249.

ISSN 1744-8050

Rubinstein, R.Y. & Kroese, D.P. (2004). *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer-Verlag.

Selman, B. & Kautz, H. (1993a). An Empirical Study of Greedy Local Search for Satisfiability Testing. In *Proceedings AAAI-93*, pages 46-51.

Selman, B. & Kautz, H. (1993b). Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems. In *Proceedings IJCAI-93*, pages 290-295.

Selman, B., Kautz, H. & Cohen, B. (1994). Noise Strategies for Improving Local Search. In *Proceedings AAAI-94*, pages 337-343.

Selman, B., Levesque, H. & Mitchell, D. (1992). A New Method for Solving Hard Satisfiability Problems. In *Proceedings AAAI-92*, pages 440-446.

Shang, Y. (1997). *Global Search Methods for Solving Nonlinear Optimization Problems*. Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, U.S.A.

Shang, Y. & Wah, B. W. (1998a). Improving the Performance of Discrete Lagrange-Multiplier Search for Solving Hard SAT Problems. In *Proceedings 10th International Conference on Tools with Artificial Intelligence*, IEEE, pages 176-183.

Shang, Y. & Wah, B.W. (1998b). A Discrete Lagrangian-Based Global-Search Method for Solving Satisfiability Problems. In *Journal of Global Optimization*, Kluwer Academic Publishers, 12, pages 61-99.

Smyth, K., Hoos, H.H., Stützle, T. (2003). Iterated Robust Tabu Search for MAX-SAT. In *Canadian Conference on AI 2003*: 129-144.

Stuckey, P. & Tam, V. (1996). Extending GENET with lazy arc consistency. Technical Report 96/8, Department of Computer Science, University of Melbourne, 3052, Parkville, Australia.

Stuckey, P. & Tam, V. (1997). Extending GENET with Lazy Constraint Consistency. In *IEEE Ninth International Conference on Tools with Artificial Intelligence (ICTAI'97)*, pages 248-257.

Stützle T. (1997). MAX-MIN Ant System for Quadratic Assignment Problems. Research Report AIDA-97-04, Department of Computer Schience, Darmstadt University of Technology, Germany.

Stützle T. (1999). Iterated Local Search for the Quadratic Assignment Problem. Research Report AIDA-99-03, Department of Computer Schience, Darmstadt University of Technology, Germany.

Taillard, E.D. (1991). Robust Tabu Search for the Quadratic Assignment Problem. In *Parallel Computing, 17,* pages 443-455.

Taillard, E.D. (1995). Comparison of Iterative Searches for the Quadratic Assignment Problem. In *Location Science, 3*, pages 87-105.

Taillard, E.D., Gambardella, L.M. (1997). Adaptive Memories for the Quadratic Assignment Problem. Research Report, IDSIA Lugano, Switzerland.

Tam, V. & Stuckey, P. (1998). Improving GENET and EGENET by new variable ordering strategies. In *International Conference on Computational Intelligence and Multimedia Applications (ICCIMA'98)*, pages 107-112.

Tsang, E.P.K. (1993). *Foundations of Constraint Satisfaction*. Academic Press.

Tsang, E.P.K. (2002). Constraint satisfaction in business process modelling, Technical Report CSM-359, University of Essex, Colchester, U.K.

Tsang, E.P.K., Wang, C.J. (1992). A Generic Neural Network Approach for Constraint Satisfaction Problems. In Taylor, J.G. (ed.), *Neural network applications*, Springer-Verlag, pages 12-22.

Tsang, E.P.K., Wang, C.J., Davenport, A., Voudouris, C., Lau, T.L. (1999). A Family of Stochastic Methods for Constraint Satisfaction and Optimization. In *The First International Conference on the Practical Application of Constraint Technologies and Logic Programming (PACLP)*, London, pages 359-383.

Tsang, E.P.K., Mills, P., Williams, R., Ford, J. & Borrett, J. (1999), A computer aided constraint programming system, *The First International Conference on The Practical Application of Constraint Technologies and Logic Programming (PACLP)*, London, pages 81-93.

Van Hentenryck, P. (2002). Constraint and Integer Programming in OPL. In *Informs Journal on Computing, 14(4)*, pages 345-372.

Voudouris, C. (1997). *Guided Local Search for Combinatorial Optimisation Problems*, Ph.D. thesis. Department of Computer Science, University of Essex, U.K.

Voudouris, C. and Tsang, E.P.K. (1998): Guided local search and its application to the traveling salesman problem. In *European Journal of Operational Research, 113,* pages 469-499.

Voudouris, C. & Tsang, E.P.K. (2003), Guided local search, in F. Glover (ed.), *Handbook of metaheuristics*, Kluwer, pages 185-218.

Wah, B.W. & Shang, Y. (1997). Discrete Lagrangian-Based Search for Solving MAX-SAT Problems. In *Proceedings of 15th International Joint Conference on Artificial Intelligence,* pages 378-383.

Wei, W., Erenrich, J. and Selman, B. (2004). Towards Efficient Sampling: Exploiting Random Walk Strategies. In *Proc. AAAI-04*, pages 670-676.

Wu, Z. (1998). The Discrete Langrangian Theory and its Application to Solve Nonlinear Discrete Constrained Optimization Problems. M.Sc. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, U.S.A.

Wu, Z. (2001). *The Theory and Applications of Discrete Constrained Optimization using Lagrange Multipliers*. Ph.D. Thesis, Dept. of Computer Science, University of Illinois, U.S.A.

Wu, Z. and Wah, B.W. (1999). Trap Escaping Strategies in Discrete Langrangian Methods for Solving Hard Satisfiability and Maximum Satisfiability Problems. In *Proceedings AAAI-99*, pages 673-678.

Wu, Z. & Wah, B.W. (2000). An Efficient Global-Search Strategy in Discrete Langrangian Methods for Solving Hard Satisfiability Problems. In *Proceedings of AAAI-2000*, pages 310-315.

Zhang, Q., Sun, J., Tsang, E. and Ford, J. (2003a). Hybrid Estimation of Distribution Algorithm for Global Optimisation, *Engineering Computations*, Accepted for publication.

Zhang, Q., Sun, J., Tsang E. and Ford, J. (2003b). Combination of Guided Local Search and Estimation of Distribution Algorithm for Solving Quadratic Assignment Problem. In *Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference*, pages 42 - 48.

Zhang, Q., Sun, J. and Tsang, E. (2004). Evolutionary Algorithm with Guided Mutation for the Maximum Clique Problem. Conditionally accepted for *IEEE Transactions on Evolutionary Computation*.