Extensions To Guided Local Search

P. H. Mills

A thesis submitted for the degree of Ph.D.

Department of Computer Science

University of Essex

2002

Abstract

In this thesis, we show how an Extended Guided Local Search can be applied to a set of problems and show that the extensions can improve its performance. We show how an aspiration criterion can be added to Guided Local Search to improve its performance for some problem types and parameter settings. We then demonstrate how, by making an occasional random move, the performance of Guided Local Search can be further improved for some problems and parameter settings. For both extensions, we make use of search monitors to attempt to analyse when and why each extension succeeds or fails. Finally, we combine the extensions and compare the resulting Extended Guided Local Search with some state-of-the-art algorithms for the different problem types, we have used for our experiments.

Acknowledgements

I would like to thank my supervisor Prof. Edward Tsang and also Dr John Ford for their guidance and encouragement to finally get me to write up my thesis, which I have no doubt that I would never have achieved by now, without it. I would also like to thank my family and friends for their support, and the many other people within the department who have encouraged me throughout my PhD.

This research was partially funded by EPSRC grants GR/L20122 and GR/M46297.

Contents

Abbreviations and Notations

ABBREVIATION	FULL NAME
GLS	Guided Local Search
SAT	Satisfiability Problem
MAX-SAT	Maximum Satisfiability Problem
QAP	Quadratic Assignment Problem
ILS	Iterated Local Search
AA	Ant Algorithms
SS	Scatter Search
GA	Genetic Algorithms
MA	Memetic Algorithms
VNS	Variable Neighbourhood Search
TS	Tabu Search
FLS	Fast Local Search
DLM	Discrete Langrangian Multipliers
GGA	Guided Genetic Algorithm
TSP	Travelling Salesman Problem
GRASP	Greedy Randomised Adaptive Search Procedure
SA	Simulated Annealing
SJ	Simulated Jumping
RTS	Robust Tabu Search
ReTS	Reactive Tabu Search
RLFAP	Radio Link Frequency Assignment Problem

GLSSAT	Guided Local Search for Satisfiability Problems
GLSMAXSAT	Guided Local Search for Maximum Satisfiability Problems
GLSQAP	Guided Local Search for Quadratic Assignment Problems

NOTATION	**DESCRIPTION**
n	Problem size (number of cities in TSP, number of elements in permutation in QAP, number of variables in SAT & MAX-SAT
D	Distance matrix for TSP
d_{ij}	Distance between cities i and j
x	A solution (a permutation in the TSP & QAP, a list of boolean variables in the SAT and MAX-SAT problems)
x_i	The ith element of the solution x
Z	The set of variables in SAT/MAX-SAT
x_i	A variable in SAT/MAX-SAT
C	The set of clauses in SAT/MAX-SAT
C_i	A clause in SAT/MAXSAT
W_{Ci}	The weight of clause C_i in MAX-SAT
A	The distance matrix in the QAP
a_{ij}	An element of the distance matrix A in the QAP
B	The flow matrix in the QAP
$b_{x_i x_j}$	An element of the flow matrix B in the QAP
$g(x)$	Original objective function
x^*	The best solution found so far, according to $g(x)$
$h(x)$	Augmented objective function used by GLS

f_i	A solution feature associated to penalties used by GLS
p_i	A penalty associated with the feature f_i
$I_i(x)$	An indicator function which defines whether solution x contains the feature f_i
$Cost_i(x)$	An cost function which gives the cost associated with feature f_i, if present in solution x
$util(x,f_i)$	The utility of penalising feature f_i in solution x
λ	The main parameter in GLS, used for altering the amount of effect the penalty term of the augmented objective function has
a	Problem specific coefficient for balancing the augmented objective function
a_{SAT}	Coefficient a for the SAT problem
$a_{MAX\text{-}SAT}$	Coefficient a for the MAX-SAT problem
a_{QAP}	Coefficient a for the QAP
v_j	A value assigned to a variable
$f_{x_i=v_j}$	The number of times $x_i = v_j$ in solutions
D_{x_i}	The set of possible values that the variable x_i may take
$N(x)$	Neighbourhood function giving neighbouring solutions of x
$f(x)$	Cost function
T	Temperature in SA
$\oplus$	Applies a move m to a solution s, returning the new solution
P_{ignore}	Probability of ignoring penalties in GLS
$P_{randmove}$	Probability of making a random move in GLS
$P_{randwalk}$	Probability of making a random walk move in GLS

$P_{randpenaltywalk}$	Probability of making a random penalty walk move in GLS
Δ	The change in some value (usually cost or augmented cost)

1 Introduction

In this chapter, we give our motivations for understanding extensions of Guided Local Search and state the scope and objectives of this thesis. We then introduce Guided Local Search, using concrete examples of Guided Local Search instantiations, which are used in the following chapters. Finally, we state and justify the experimental methodology that we use throughout the rest of the thesis.

1.1 Motivations

Guided Local Search (GLS) [102] has been applied to a number of problems, including the vehicle routing problem [47], the radio link frequency assignment problem [104], function optimization [101] and the travelling salesman problem [103]. In this thesis, we use the SAT, weighted MAX-SAT and QAP problems as testbeds (see chapter 2 for more details) for our extensions of Guided Local Search.

GLS is a meta-heuristic that sits on top of local search procedures and helps them escape from local minima. GLS can be seen as a generalization of the GENET neural network [95,16,20] for solving constraint satisfaction problems and optimization problems. Recently, it has been shown that GLS can be put on top of a specialized Genetic Algorithm, resulting in the Guided Genetic Algorithm (GGA) [50]. GGA has been applied to a number of problems, including the processor configuration problem [52], the generalized assignment problem [51] and the radio link frequency assignment problem [52,53].

In this thesis, we examine ways of extending Guided Local Search, so that the performance can be improved for some problems and/or parameter settings, whilst

hopefully having a minimal degradation (if any) on performance for other problems/parameter settings. We believe it is important to examine these possible extensions in a systematic and intelligent manner, owing to the vast number of possible ways of extending GLS. Later in this chapter, we outline our methodology for achieving this end.

1.2 Scope and objectives

In this thesis, we focus on:

- extending Guided Local Search only, and not attempting to modify other algorithms,
- enhancing the parameter setting range of GLS, or at least improving its performance for some settings and/or problems, and
- understanding what effect the extensions have on the search algorithm, and hopefully therefore gaining an idea of why they work, when and for which problems types.

1.3 Problems

In this section, we give examples of problems commonly studied by researchers in discrete optimization and local search metaheuristics.

1.3.1 The Travelling Salesman Problem (TSP)

The travelling salesman problem [55] is probably the most famous combinatorial optimisation problem and one of the most heavily researched. Because of the simplicity of the basic symmetric TSP and the way it lends itself to graphical representation of solutions, it is an ideal candidate for illustrating how general algorithms may be tailored to a particular problem.

The problem is to find a tour (a list or permutation of n cities) through a set of cities, such that the length of the tour is minimised. The problem can be defined by a matrix D, each element d_{ij} of which defines the distance between a city i and a city j. The problem is then to find a permutation of n cities, such that the sum of the distances between each city in the permutation is minimised. This can be stated formally as:

$$\min_{x}\left(\left(\sum_{i=0}^{n-1} d_{x_i x_{i+1}}\right)+d_{x_n x_0}\right) \quad (1)$$

where:

D is a matrix whose elements $d_{x_i x_j}$ represent the distance between cities x_i and x_j

x is a permutation representing the order in which the n cities are visited.

1.3.2 The Satisfiability Problem (SAT)

The SAT problem [31] is an important class of constraint satisfaction problem [94], where the domain of a variable is always the set *{false, true}*, and each constraint is a logical disjunction. The SAT problem is important in solving many practical problems in mathematical logic, constraint satisfaction, VLSI engineering and computing theory [35].

The SAT problem is defined below:

- A set of n (boolean) variables, $Z = \{x_1, x_2, .., x_n\}$, each of which may take the values true or false.
- A set of m clauses, $C = \{C_1, C_2, .., C_m\}$, each of which is a disjunction of a set of literals (a variable or its negation), e.g. $x_1 \vee \neg x_2 \vee x_3$.
- The goal of the SAT problem is to find an assignment of values to variables (if one exists) where all the clauses are satisfied (evaluate to true) or prove that the problem is unsatisfiable if no valid assignment exists (currently, only complete algorithms can prove unsatisfiability).

Both complete and incomplete algorithms have been used to solve the SAT problem. Of the complete algorithms, one of the best known is the Davis-Putnam procedure [21], based on resolution. Of the incomplete local search algorithms, the best known is probably GSAT (first reported in [81]), based on random restarts and steepest gradient descent and the related WalkSAT [79], based on random walk with greedy variable selection heuristics. Another penalty-based algorithm called DLM [84, 83, 82, 113, 115] has also been applied to the SAT problem with good results.

When local search is applied to the SAT problem, it is well known to contain many long plateaus of so-called sideways moves (moves to solutions of identical cost), as shown in [77] and [27]. This means that the algorithm is stuck in a "flat" landscape of these solutions, moving between solutions of equal cost, with nothing to guide it as to which direction to move next. This moving between solutions of equal cost whilst traversing the plateau may carry on until some solution is found with neighbouring solutions of lower cost, usually leading to an even larger "plateau" of solutions of slightly lower cost (this carries on until the lowest plateau is reached (and the search continues indefinitely if no meta-heuristic is applied), or a solution which satisfies all the clauses is obtained).

1.3.3 The Weighted MAX-SAT Problem

The weighted MAX-SAT problem [75] is defined as follows:

- A set of n variables, $Z = \{x_1, x_2, \ldots, x_n\}$, each of which can take the values true or false.
- A set of m clauses, $C = \{C_1, C_2, \ldots, C_m\}$, each of which is a disjunction of a set of literals (a variable or its negation), e.g. $x_1 \vee \neg x_2 \vee x_3$. Each clause has a weight W_{C_i} associated with it.

- The goal of the weighted MAX-SAT problem is to find an assignment of values to variables, if one exists, where all the clauses are satisfied (evaluate to true) or, if none exists, find an assignment that maximizes the sum of the weights of satisfied clauses.

Only a small amount of work has been done on the weighted MAX-SAT problem in comparison to the pure SAT problem. For example, [43] extends GSAT to handle the weighted MAX-SAT problem, and has had some success in solving instances based on the network Steiner tree problems, in comparison to some techniques specialised for the task. Another algorithm, GRASP [75], has also been applied to a set of weighted MAX-SAT problems, this time derived from a set of problems from the DIMACS archive. Recently, in [107] a new and effective discrete Langrangian-multiplier-based global-search method, called DLM, has been applied to the weighted MAX-SAT problem. This produces excellent results on the same set of benchmark instances as in [75].

The landscape of the weighted MAX-SAT problem is similar to that of the SAT problem, if the number of unsatisfied clauses is used as the objective function. If the weighted sum of unsatisfied clauses is used (the natural choice of objective function), the landscapes may differ, however.

1.3.4 The Quadratic Assignment Problem

The Quadratic Assignment Problem (QAP) [8] is one of the hardest groups of problems in combinatorial optimization, with many real-world applications, and it has been the focus of much successful research into heuristic search methods.

Roughly speaking, the QAP can be thought of as trying to find a suitable plan for placing a set of facilities at a set of given locations, where we know the quantities of

objects which flow between the different facilities and the distances between the different location. The problem is defined by two matrices *A* and *B*. Element a_{ij} of the matrix *A* gives the distance between the different locations labelled *i* and *j*. An element $b_{x_i x_j}$ of the matrix *B* gives the flow between the facilities x_i and x_j. The problem is to find a permutation *x* (representing which facility x_i is placed at location *i*) which minimizes the sum of the distance times the flow between the different facilities. The problem can be formally stated as:

$$\min_x \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{x_i x_j} \tag{2}$$

where:

- *n* is the size of the problem (i.e. number of facilities or locations);
- *x* is a permutation of (*1*, *2*, .. , *n*) and where x_i is the i^{th} element in permutation *x*, representing the facility placed at location *i*;
- *A* is the distance matrix, each element a_{ij} representing the distance between locations *i* and *j*;
- *B* is the flow matrix, each element $b_{x_i x_j}$ representing the flow between facilities x_i and x_j.

Both exact and heuristic (inexact) algorithms have been proposed for solving the Quadratic Assignment Problem (for a survey, see [69]). The exact algorithms have the disadvantage that they can only find optimal solutions to relatively small QAPs (where $n <= 20$), whereas the heuristic methods can find near-optimal if not optimal solutions (it is often unknown if these are the optimal solutions, since no exact or complete methods exist which can solve them in a reasonable amount of time) to much larger problems (where $n > 20$). The heuristic methods which have been used to solve the QAP include Robust Tabu Search [89, 90], Reactive Tabu Search [4],

Simulated Annealing [93, 111], Simulated Jumping [1], a Memetic Algorithm [60], a Genetic Hybrid Algorithm [24], Ant Algorithms [29, 87] and others [91]. In this thesis, we show how Guided Local Search can be applied to the Quadratic Assignment problem, and present empirical results showing two extensions of Guided Local Search which can increase the range of parameters under which good results are obtained.

Some studies of a measure of the ruggedness of the search landscape called the ruggedness coefficient have been carried out for the QAP [61, 2, 3].

1.4 Guided Local Search

Guided local search (GLS) (see [102] for a more detailed description) 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 search 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.

1.4.1 Local Search

For each problem, a neighbourhood function *N(x)* which returns a set of neighbouring solutions to *x* and an objective function *g(x)* giving the cost of the solution *x*, must be defined for the local search algorithm (see Section 2.1 for a description of local search) used by GLS. For the SAT problem, the neighbourhood *N(x)* is simply the set of all solutions resulting from changing the value of one variable from true to false or vice-versa. The objective function *g(x)* is the number of unsatisfied clauses, given the solution *x*. For all problems, the standard local search algorithm (see Figure 1-1 for

pseudo code) we use for GLS, allows a maximum of two consecutive sideways moves before the local search algorithm terminates. We use the same objective function and neighbourhood scheme for the MAX-SAT problem (although the weight W_{Ci} for each clause C_i is used by GLS, see Section 1.4.2 and the sum of the weights of unsatisfied clauses is used to record the best found solution x^*) as for the SAT problem. For the Quadratic Assignment Problem, the neighbourhood function $N(x)$ is set of all solutions with two distinct elements of the permutation swapped (see [5,89,90] for details of how to update the change in objective function of the neighbourhood efficiently). The objective functions used (this is the original objective function denoted as $g(x)$ later and used in the augmented objective function $h(x)$, defined later) for the SAT and weighted MAX-SAT[1] problems are to minimise the number of unsatisfied clauses, whilst for the QAP $g(x)$ is as defined earlier in (2) in Section 1.3.4.

1.4.2 Solution features

Solution features are defined to distinguish between solutions with different characteristics, so that poor characteristics can be penalized 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 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

[1] This is not the natural formulation for the weighted MAX-SAT problem, but this has been used successfully for problems involving soft constraints before, e.g. for the RLFAP in [104] and the weighted MAX-SAT in [64].

each feature f_i an Indicator Function I_i indicating whether the feature is present in the current solution or not:

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

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

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

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

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

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

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

1.4.3 Selective penalty modifications

When the Local Search algorithm returns a local minimum, *x*, GLS penalizes all those features (through increments to the penalty of the features) present in that solution which have maximum utility, $util(x,f_i)$, as defined in (6). (See Figure 1 for pseudo code of the overall GLS algorithm.)

$$util(x, f_i) = I_i(x) \cdot \frac{c_i(x)}{1 + p_i} \quad \textbf{(6)}$$

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

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

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

          return x
}
```

where:

- *g(x)* returns the cost of a solution *x* with regard to the original cost function,
- *h(x)* returns the augmented cost of a solution *x*,
- *x** is the solution of lowest (original) cost found so far by the algorithm,
- *N(x)* is the neighbourhood function, giving neighbouring solutions of *x*. See Section 1.4.1 for problem specific neighbourhood functions used in this thesis.

Figure 1-1: Pseudo code for GLS

1.4.4 Augmented cost function

GLS uses an augmented cost function (7), to allow it to guide the Local Search algorithm out of the local minimum, by penalizing 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.

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

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 in the search 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 with finer steps. 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. However in this thesis *a* is pre-defined based on each problem (8), as below. This is because the former method may depend on the initial solution, whereas a fixed value will make the experiments more easily repeatable.

$$a_{SAT} = 1$$

$$a_{MAX\text{-}SAT} = 1 \quad (8)$$

$$a_{QAP} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} \times \sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij}}{n^4}$$

Summarising, to use GLS, a number of things need to be defined: an objective function, a local search algorithm, a set of features found in solutions together with indicator and cost functions, and also the *a* coefficient and the *λ* intensity/diversity setting.

1.5 Methodology

In this section, we outline the experimental methodology that we use in the rest of this thesis.

1.5.1 Empirical evaluation with search monitors

In this thesis, we evaluate and test our extensions of GLS empirically, using search statistics (which we call "search monitors") collected over multiple runs of Guided Local Search. Table 1.1 defines the search monitors which we use in this thesis (although it should be noted that many others were tried, but those below were found to be the most useful for our purposes).

Name of search monitor	**Definition**	**Purpose**
Repetitions	Number of times solutions are revisited	Monitors the extent to which the search algorithm revisits previous solutions.
Average label entropy	The average label entropy (for a description of entropy see [7]) is defined below: $Average\_label\_entropy = \frac{\sum_{i=0}^{n} Entropy(x_i)}{n}$ where $Entropy(x_i)$ of a single variables x_i in the problem is defined below:	Monitors how diverse the search is.

	$Entropy(x_i) = \sum_{\substack{v_j \in D_{x_i} \\ f_{x_i=v_j} \neq 0}} -\frac{f_{v_j}}{iterations} \cdot \frac{\log\left(\frac{f_{x_i=v_j}}{iterations}\right)}{\log(\lvert D_{x_i} \rvert)}$ where x_i is a variable, D_{x_i} is the set of possible values of variable x_i, v_j is a value of the variable x_i & $f_{x_i=v_j}$ the number of occurrences of the label $x_i=v_j$ in solutions visited by the search so far, *iterations* is the number of solutions visited by the local search algorithm so far.	
Average solution cost	This is the average cost of all solutions visited over a run.	Monitors the quality of solutions visited over the whole search.
Average local minimum cost	This is the average cost of all local minima found over a run.	Monitors the quality of local minima found during the search.
Average better-than previous Solution cost	This is the average cost of all solutions whose cost is better-than-previous found during a run.	Monitors the quality of best found solutions found over a whole run.

Table 1: Search monitors for GLS

1.5.2 Benchmark problems

During our empirical tests, we use benchmarks found by other researchers, as these are known to contain "hard problems", yet still contain a sufficient variety and number of problems to be of interest.

1.6 Conclusion

In this chapter, we have introduced our motivations for extending Guided Local Search. Finally, we have introduced our experimental methodology, which is designed to assess empirically the effects of the extensions of Guided Local Search, using search statistics, which we call "search monitors" collected over multiple runs of Guided Local Search.

2 Related work

In this chapter, we examine other algorithms which perform a similar function to Guided Local Search.

2.1 Local Search

Local search algorithms work by starting with a solution (usually randomly generated) representing some possible configuration (for example, a tour of cities in the TSP) and then making small changes to that solution that decrease the cost of the configuration. A local search algorithm must consist of a *neighbourhood function* for generating the set of neighbouring solutions *N(x)* of a solution *x, a cost function* for evaluating the cost *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 that very often after a few *"moves"* (small changes to solutions) to neighbouring solutions, the cost function *f(x)* can no longer be reduced (if minimising) and the algorithm is stuck in what is known as a *local minimum*.

2.2 Metaheuristics

Meta-heuristics are heuristics that are designed to control heuristics like local search (in this thesis) to either avoid or escape from the local minima described in the previous subsection. Many such metaheuristics exist, each with many variations on the theme and, due to this, we restrict our attention to the main ones. In Figure 2-1, we show an approximate map of current metaheuristic research related to Guided Local Search (we recognise that this is by no means a complete picture or a unified view of current research).

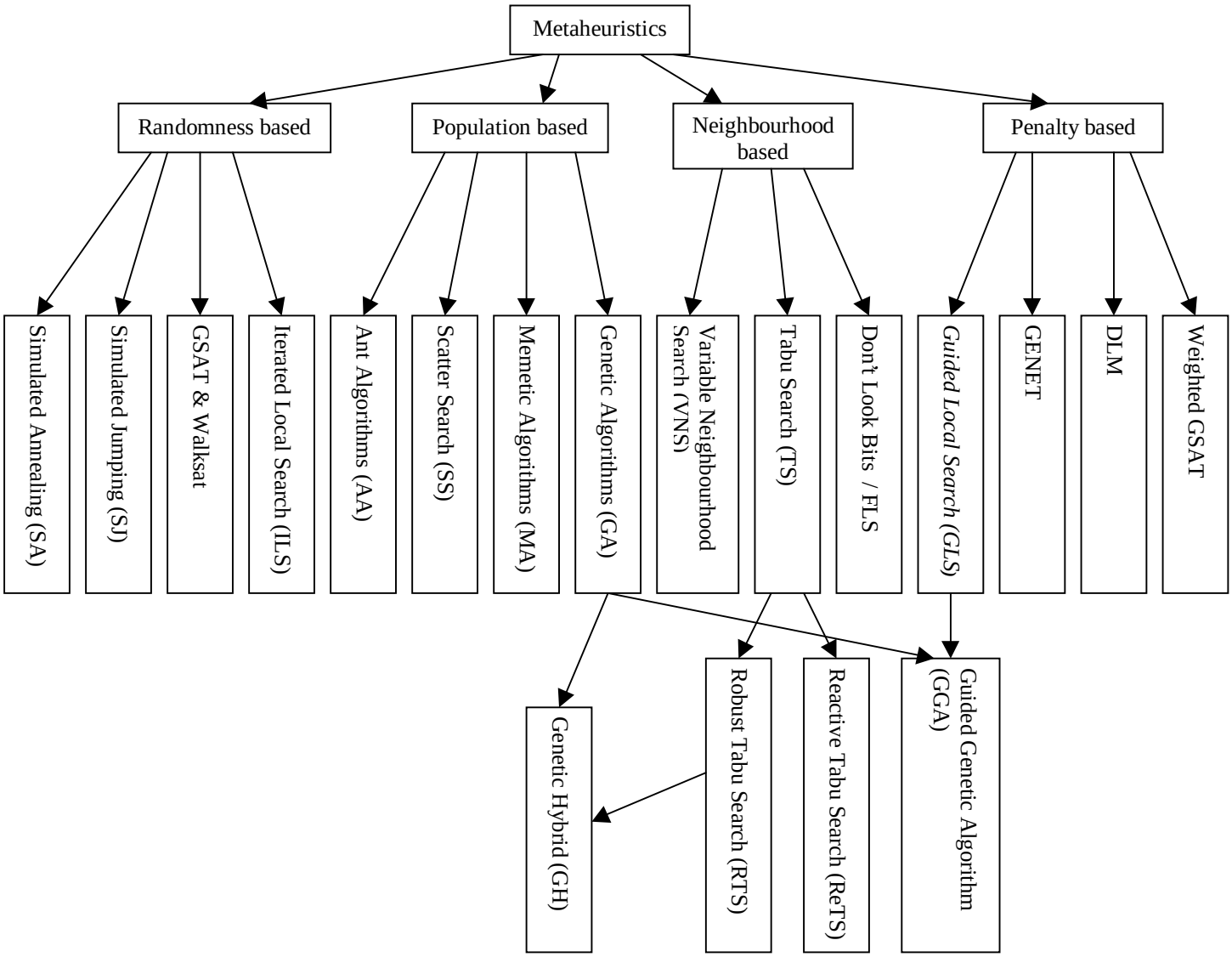


Figure 2-1: An approximate map of the position of Guided Local Search (GLS) in metaheuristic research

We split algorithms into four main classes: those using some element of randomness 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 multiple solutions in parallel, rather than just concentrating on a single solution, neighbourhood-modification based metaheuristics and those using 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.

2.3 Randomness based meta-heuristics

In this section, we describe other meta-heuristics that perform a similar function to Guided Local Search, but which do so by adding some form of randomness or noise to the search process.

2.3.1 Simulated Annealing

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

Figure 2-2: Pseudo code for Simulated Annealing

Simulated Annealing [48, 60] 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, instead forming a solid whose molecules will not be in a

minimum energy state. The idea is that the moves made by an iterative improvement algorithm are like the re-arrangements of molecules in a liquid that occur as it is cooled and that the energy of those molecules is like 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, by making downwards moves with occasional "upwards" moves (the probability of these occurring decreasing with the "temperature") and by doing this hopefully it ends 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 global optimum solution.

Figure 2-2 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, assuming we are minimising) 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 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 terminates when some termination condition becomes satisfied (typically when no improvement has been

made for a certain number of iterations or the maximum number of iterations has been reached).

2.3.2 Simulated Jumping

A variation on Simulated Annealing is Simulated Jumping [1] which is a relatively new meta-heuristic. It is based on the idea in physics that some materials containing both ferromagnetic and anti-ferromagnetic materials 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 in combinatorial optimisation problems. Rather than gradually decreasing over a run the probability of accepting an upwards (if we are minimising) 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 2-3. The cooling and heating schedules are those suggested in [1] 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.

```
SimulatedJumping(T0, γ, R, MaxCycles)
{
      //Typical values for parameters: (from [1])
      //T0 = 0.001, γ = [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
            Δg = g(y) - g(x)
            if (Δg < 0) x = y
            else
            {
                  r = random number in range [0,1]
                  if (r < e^(-Δg/T))     x = y       //Accept the change
                  else                   T = T+R/i   //Heat the system
            }
            T = γ * T //Cool the system
            if (g(x) < g(x*)) x* = x
         }
      }
      while not termination condition
}
```

Figure 2-3: Pseudo code for Simulated Jumping

2.3.3 GSAT and Walksat

GSAT [81, 77, 78] and Walksat [79, 59] are algorithms for dealing specifically with the SAT problem (a version of Walksat was also adapted for solving weighted MAX-SAT problems [43]). 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 2-4. 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*, flipping that variable which decreases the maximum number of unsatisfied clauses, or with probability *noise* flipping a randomly picked variable that is involved in one or more unsatisfied clauses), 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 the 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 2-4: Pseudo code for GSAT

Pseudo code for the basic Walksat is given in Figure 2-5. 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 clause that becomes unsatisfied from satisfied 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 [78,79,45,46,10,26,25,27,43] 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 the variable in c which minimises
                          the number of breaks flipped (clauses which
                          become unsatisfied from satisfied as a
                          result of the flipping the variable)
                  }
                  j = j + 1
            }
            if (#unsat_clauses = 0) return x

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

Figure 2-5: Pseudo code for walksat

2.3.4 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 Local Search [88] meta-heuristic. The main motivation behind this approach is to utilise the previous local minima solutions and recombine (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) to create new start points in order to increase the amount of time exploring more promising regions of the search space.

The mutations of previous local optima are commonly known as kick-moves, and 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
      {
            x = Modify(x,history)
            y = LocalSearch(x)
            x = AcceptanceCriterion(x,y,history)
      }
      while (termination condition not met)
}
```

Figure 2-6: Pseudo code for Iterated Local Search

Pseudo code for ILS is given in Figure 2-6. 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. The Modify function then 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 then improved by the local search until a local minimum y is found and returned. 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 s 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, however, are in the way in which Modify changes the best found solution so far and how large (how many random moves it is composed of) the "kick-move" is.

2.4 Population based algorithms

In this section, we describe meta-heuristics that store a population of solutions to help prevent them converging prematurely.

2.4.1 Ant algorithms (AAs)

Ant algorithms [22, 29, 87, 91] are based on the idea of having a population of solutions, with each solution worked on by an individual "ant", with all the ants sharing a common data structure containing "pheromone information" accumulated over the course of the search. For example, in the TSP [22], the pheromone information is a matrix, representing the amount of pheromone on each edge joining two cities. The higher the value for the pheromone trail on each edge, the more desirable the edge is.

Pseudo code for an ant algorithm, based on the HAS-QAP from [29], is given in Figure 2-7. 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 of either exploitation (attempting to improve the current ant's solution [partly greedily, according to the pheromone information, and partly randomly]) or exploration (attempting to modify the current ant's solution, partly randomly and partly probabilistically weighted towards solutions containing high amounts of pheromone). The resulting solution from the modifications is then improved using local search. If the algorithm is in an intensification phase, 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" 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(γ,α1,α2,q,R,S) //e.g. γ=100, α1=0.1, α2=0.1, q=0.9, R=n/3
{
      foreach ant i
      {
            x_ant_i = GenerateInitialStartPoint()
            x_ant_i = LocalSearch(x_ant_i)
      }
      foreach pheromone trail j, T_j = 1 / (γ.g(x*))
      intensification = true

      do
      {
            foreach ant i = 1 to n
            {
                for k = 1 to r
                {
                  withprobability (q)
                  {  //perform exploitation
                     choose a neighbouring solution x_ant_i^k from N(x_ant_i)
                     partly randomly, and partly such that the amount
                     of pheromone is maximised
                  }
                  else
                  {  //perform exploration
                     choose a neighbouring solution x_ant_i^k from N(x_ant_i)
                     partly randomly and partly randomly weighted
                     towards those solutions with high amounts of
                     pheromone
                  }
                }

                x_ant_i^(r+1) = LocalSearch(x_ant_i^r)

                if intensification = true
                  x_ant_i = best x_ant_i^k for k = 1 to r+1
                else
                  x_ant_i = x_ant_i^k
            }

            if no improvement in any solution x_ant_i this cycle
                  intensification = false
            if there exists an x_ant_i, such that g(x_ant_i) < g(x*)
            {
                  x* = x_ant_i
                  intensification = true
            }

            //Evaporation
            foreach pheromone trail T_j = T_j .(1-α1)

            //Reinforcement
            foreach pheromone trail T_b present in solution x*
                  T_b = T_b + α2/g(x*)

            if S iterations have past, without x* being improved
            {     //Perform diversification
                  foreach ant i, x_ant_i = GenerateInitialStartPoint()
                  foreach pheromone trail j, T_j = 1 / (γ.g(x*))
                  x_ant_1 = x* //Keep the best solution so far
            }
      }
      while not termination condition
}
```

Figure 2-7: Pseudo code for an ant algorithm (based on HAS-QAP from [29])

2.4.2 Scatter Search (SS)

Scatter Search [14] is a simple evolutionary heuristic which is very similar to memetic and genetic algorithms. Pseudo code for Scatter Search is given in Figure 2-8. 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 *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 *Q* 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 2-8: Pseudo code for Scatter Search

2.4.3 Genetic Algorithms (GAs)

Genetic Algorithms [41] 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 to solve a problem, a cost function must be defined for evaluating potential solutions along with a suitable representation for those solutions, along 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. Pseudo code for a basic Genetic Algorithm is given in Figure 2-9. The algorithm starts by creating an initial population of solutions, and then creating a new generation, by means of probabilistically selecting parents and individuals to perform crossover, mutation and reproduction a number of times until the new population has reached the predefined population size. This process then continues until some termination condition is reached (e.g. a sufficiently good solution has been found or the algorithm has performed the maximum number of iterations has been reached or no improvement has been made for a number of iterations).

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 [24], 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.

```
GeneticAlgorithm(Pr,Pc,Pm)
{
      population = GenerateInitialPopulation()

      do
      {
            next_population = {}

            for i=1 to population_size
            {
                  random_number = random number between [0,1]
                  if (random_number < Pr) //reproduce
                  {
                     solution = roulettewheelselection(population)
                     next_population = next_population ∪ {solution}
                  }
                  else if (random_number < Pr+Pc) //crossover
                  {
                     sol1 = roulettewheelselection(population)
                     sol2 = roulettewheelselection(population)
                     child = crossover(sol1,sol2)
                     next_population = next_population ∪ { child }
                  }
                  else //mutation
                  {
                     solution = roulettewheelselection(population)
                     mutated_solution = mutate(solution)
                     next_population = next_population ∪
                                              {mutated_solution}

                  }
            }

            population = next_population
      }
      while (not termination condition)
}
```

Figure 2-9: Psuedo code for a basic Genetic Algorithm (adapted from [49])

2.4.4 Memetic algorithms (MAs)

Memetic algorithms [67,68] 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 and 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 from population randomly
                  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 2-10: Pseudo code for an example of a simple Memetic Algorithm (MA) (adapted from [60])

Figure 2-10 shows pseudo code for a simple example of a memetic algorithm, which 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. Then a local search algorithm is again applied to the resulting meme, which is 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 [24], already covered in the previous section on Genetic Algorithms. These methods are also very similar to the elite solutions restarting [40] from tabu search, 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 [23].

2.5 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.

2.5.1 Variable Neighbourhood Search (VNS)

Variable Neighbourhood Search (VNS) [65] has several local search neighbourhoods of increasing size available to it. It starts at some initial start point (usually randomly chosen). It then picks a neighbouring solution at random from the smallest-sized neighbourhood and then 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 2-11.

```
VariableNeighbourhoodSearch(N1(),N2(),..,Nkmax())
{
      x = GenerateInitialSolution()
      k = 1
      do
      {
            y = random solution picked from Nk(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
            {
                  k = k + 1 //=>use next largest neighbourhood
            }
      }
      while (not termination condition)
}
```

Figure 2-11: Pseudo code for basic Variable Neighbourhood Search (VNS)

2.5.2 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 (g(y) < g(x*))
         if (g(y) < g(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 2-12: Pseudo code for basic Tabu Search with the best-improved aspiration criterion

Tabu Search [37, 38, 39, 40] 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 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 2-12.

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, I believe that while there may be some element of truth in this, there is also an element that if one tries hard enough, it is always possible to draw parallels between different search methods. For this reason, I 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's book [40] for a detailed look at the many ideas that he and other researchers have had.

2.5.2.1 Robust Tabu Search (RTS)

Robust tabu search is an enhanced version of the basic tabu search scheme, which uses a randomly varing 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, [89,90]) around some fixed value (the number of elements in a solution permutation in the QAP, [89,90]) every time a local search move is made. The long term memory forces solution attributes back into solutions 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). This is done by making any move which does not introduce the desired attribute "tabu" and disallowed (unless the best-improved aspiration criterion is applicable). Robust Tabu Search has been shown to be successful in tackling the Quadratic Assignment Problem (QAP).

2.5.2.2 Reactive Tabu Search (ReTS)

Reactive Tabu Search (ReTS) 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 [4,5]. 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 thing that Reactive Tabu Search does is to make 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 a number of solutions are revisited) which for some reason cannot be escaped from just using a tabu search strategy. This part of ReTS resembles the idea of Iterated Local Search (ILS), where a similar method is employed to escape from local minimas, rather than using a random restart from a completely new solution.

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

Fast Local Search (FLS) [102] is an generalisation/adaptation of an earlier scheme known as *"don't look bits"* [6] 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 (RLFAP)) and recently, the Quadratic Assignment with Iterated Local Search [88].

```
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' effected 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 2-13: 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 (if minimising) 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 2-13 shows pseudo code giving a rough 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 [39,40]. 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 this 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 neighbourhood is massive, these techniques may make a large saving in running time.

2.6 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 optimizing in order to help them escape from local minima.

2.6.1 Guided Local Search

For more details of Guided Local Search, refer to the section on it in Chapter 1, or [102].

2.6.2 GENET and other Weighted constraint algorithms

GENET (the algorithm which Guided Local Search generalised to the wider group of combinatorial optimisation problems [98,99,102]) 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 to what degree 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 that 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* of the GENET algorithm for historical reasons (GENET was originally a Neural Network)).

Pseudo code for the basic limited sideways GENET scheme is shown in Figure 2-14. 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 like this until some termination condition is satisfied or all the constraints are satisfied.

For more information on GENET the interested reader may refer to [20]. For a historical interest in GENET's development, the interested reader may refer to [95,94,16,97]. Much work has been carried out on extending GENET. This includes

adding lazy constraint consistency to GENET [85,86] and introducing variable ordering strategies [92] to attempt to improve its performance, as well as various other schemes based on adding additional constraints and "nogood" constraints to the problem [56,57,58]. For a study of GENET compared to Tabu Search on a group of partial constraint satisfaction problems, the interested reader should refer to [9]. GENET has also been shown to belong to the class of Discrete Langrangian Search algorithms in [12].

```
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_xi
      foreach c_i in C, w_ci = -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_xi such that it maximises sum of
                     V_ci(x_0..x_n)*w_ci of violated constraints on x_i
                     (Break ties randomly)

               if (sum(V_ci(x_0..x_n)*w_ci) 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_ci = w_ci - 1
      }
      if (#{c_i in C | c_i is not satisfied} = 0)
            return true       //solution found
      else
            return false      //no solution found
}
```

Figure 2-14: 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 [66], 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 in [66] 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 (see section 2.3.3 in this chapter) with weights [77,78]. 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 [25]. Later, this scheme was extended to also decrease all weights of clauses after each flip as well [26], although we believe this "short term" weighted clause approach is probably infeasible owing to excessive CPU requirements of decreasing *every* clause's weight after *every flip* of a variable.

2.6.3 Discrete Langrangian Multipliers (DLM)

The Discrete Langrangian Multiplier (DLM) search algorithm is based on a modified mathematical theory from continuous optimisation. DLM associates a "Langrangian multiplier" with each constraint in the problem, which is increased each time DLM reaches a local minimum. As one can easily see, this is almost exactly what GENET does, and in fact GENET has been shown to belong to the same class of Langrangian-based search algorithms in [12].

DLM has been applied to a number of problems, including the weighted MAX-SAT problem [107], the SAT problem [84], as well as others, such as some continuous and

mixed integer programming problems [82]. The theory is that the algorithm is maximising the Langrangian multipliers (performing ascent in the Langragian 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 [83]. An ad hoc "trap" escaping strategy has been added to DLM to improve its performance in [115]. This strategy increases the Langrangian multipliers more than usual for clauses, that become unsatisfied more frequently. However, this trap escaping strategy has no mathematical basis, although it has produced very good results on hard SAT benchmark problems. A slightly less "ad hoc" and more general scheme, which performs the same job as the "trap escaping" strategy, is given in [116], 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 [82, 113, 114].

2.6.4 Tabu Search Penalties

Tabu Search (see earlier section), 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 [23]. 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* [40] 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).

3 An Aspiration Criterion for Guided Local Search

Aspiration criteria have been commonly used with tabu search algorithms (see [40] for a description of the many types of aspiration criteria used in tabu search). In this chapter we give our motivation for adding a simple aspiration criterion to Guided Local Search. We show how Guided Local Search can be extended using an aspiration criterion. We then present experimental results on three groups of problems: the SAT problem, the QAP problem, and the weighted MAX-SAT problem. Finally, we put forward some theories, based on our experimental results, as to when and why this extension may improve the performance of Guided Local Search.

3.1 Motivation

Guided Local Search (see chapter 1) is a penalty-based method, whereby higher cost solution features found in local minima are penalised until the local search algorithm can escape from the local minimum solution. However, sometimes a feature may be penalised early on in the search and then at a future point during the search, this penalty may become less relevant and/or misleading to the local search algorithm in the current part of the search space, either because the cost of the penalised feature has changed (e.g. this happens in the QAP), or because the penalised feature acts as a barrier in the search space, preventing GLS from visiting some solutions. This is particularly damaging if an area of the search space is visited where there is a possibility of finding a new best solution, but where this new best solution may be prevented (by penalties previously imposed in the search) from being visited. This blocking of solutions is made more likely if a higher value of λ is used, as this increases the influence of penalties in the augmented objective function, whereas a

lower value of λ decreases the chances of this situation happening, but reduces the effect of the penalties. Thus, if some method for avoiding this situation can be found without having to take into account the λ value, then GLS should be less sensitive to the λ parameter and also should be better able to deal with problems where the cost of features varies during the search.

3.2 Definition of an aspiration criterion

Aspiration criterion is an idea that comes from Tabu Search (see [37]). In tabu search, an aspiration criterion is any condition under which the status of a tabu move or a tabu attribute may be overridden. The most commonly used form of aspiration criteron is called the *improved best* aspiration criterion [40, p36] whereby if a new improved solution can be obtained by a tabu move, then the tabu status of that move is ignored and the move is executed anyway, thus obtaining a new best solution. In this chapter we only consider the *improved best* aspiration criterion, unless otherwise stated. Several other types of aspiration criterion exist, and the interested reader may refer to [40].

In Guided Local Search, we have a set of penalties imposed on solution features, rather than a list of tabu solution attributes and/or a list of tabu moves. Thus in Guided Local Search, an *(improved best) aspiration move* is defined to be a move such that a new best found solution is generated by that move, and that move would not have otherwise been chosen by the local search using the augmented objective function. Pseudo code for local search with aspiration moves for GLS is given in Figure 3-1.

```
Local_Search_With_Aspiration(x,h,g,N)
{
  do
  {
    if (Get_Aspiration_Move(x,h,g,N,z))
      x = z
    else
    {
      y =  y in N(x) such that h(y) is minimised,
           breaking ties randomly

      Δh = h(y) - h(x)

      if (Δh <= 0) x = y
      if (Δh = 0)  sideways = sideways + 1
      else         sideways = 0
    }

    if (g(x) < g(x*)) x* = x
  }
  while (Δh <= 0) and (sideways < 2)

  return x
}

Get_Aspiration_Move(x,h,g,N,z)
{
  z = z in N(x) such that g(z) is minimised, breaking ties randomly
  if (g(z) < g(x*) and ((h(z)-h(x)) > 0))
     return true
  else
     return false
}
```

where:

- *x,y & z* are solutions,
- *g()* returns the cost of a solution with regard to the original cost function,
- *h()* returns the augmented cost of a solution,
- *x** is the solution of lowest (original) cost found so far by the algorithm,
- *N(x)* is the neighbourhood function, giving neighbouring solutions of x.

Figure 3-1: Pseudo code for local search for GLS with aspiration moves

This may be simply implemented by considering, first of all, if there is any move in the neighbourhood that will yield a new best solution, according to the cost function *f*. The neighbourhood is then examined using the augmented objective function *g* to see if a move exists which will reduce the augmented cost of the next solution. If the best move in the neighbourhood w.r.t. the original objective function yields a new best

solution, whose original objective function cost is less than that of the move that would have been chosen according to the augmented objective function, then we choose that move which generates the lowest cost new best solution (this is deemed to be an aspiring move), breaking ties randomly. Otherwise, we choose the move with the lowest augmented objective function cost, breaking ties randomly.

3.3 Experiments

We ran experiments on the SAT, weighted MAX-SAT and Quadratic Assignment problems, using both the standard GLS schemes for those problems (see Chapter 1) and these GLS schemes along with the improved best aspiration criterion.

For the SAT problem, we ran both algorithms, allowing a maximum of 10**n* repairs (where *n* = number of variables in the problem) and 10 runs, from random start points on the easier[2] soluble problems (129 problems in all) in the DIMACS benchmark [44] suite[3]. These were all the problems in the set {aim*, as*, ii*, jnh*, par8-?-c, ssa*, tm*}. For detailed information about the number of variables and clauses for each problem, see the Appendix. We chose not to run on the harder (those problems which took many hours or days to solve) problems, since it would have been too costly, in terms of CPU run time, to perform all the experiments we wanted to perform. We varied the λ parameter over the set of values {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100}.

For the weighted MAX-SAT problem we ran GLS with and without aspiration on the problems (44 problems in all) from [75]. For detailed information about the number of variables and clauses, see the Appendix. We allowed a maximum of 10**n* repairs

and 10 runs for each problem, over the same set of λ parameter values as for the SAT problem.

For the QAP problem, we ran GLS with and without the aspiration criterion over all the small to medium size[4] problems (i.e. where $n < 50$; for a list of problems used and their size, see the Appendix) in the QAPLib [8], allowing a maximum of 1000*n repairs (we allowed more repairs for the QAP problems, since they in general require more repairs to find the best known solution, than the SAT or MAX-SAT problems) with 10 runs for each problem and the same set of λ coefficients as the set used for the SAT problem.

In addition to this, we also ran a control experiment where, with probability P_{ignore}={0.2, 0.4, 0.6, 0.8}, we allowed a move to be made according to the original objective function. We used the same problems, and the same number of repairs and runs as for the other experiments above, whilst performing each experiment with λ coefficients in the set {0.1, 1, 10, 100}. This was to test the hypothesis that just ignoring penalties every so often might yield the same effect as the best-improved aspiration criterion, or at least account for some of the effect of it.

3.3.1 Search monitors used for aspiration move experiments

During all these experiments, we recorded information gathered by the search monitors (see Chapter 1), to help us evaluate what was happening during the search. The search monitors we measured specifically for this chapter were:

[2] We only used the easier problems due to time constraints.
[3] DIMACS benchmarks available from ftp://dimacs.rutgers.edu/pub/challenge/sat/benchmarks/cnf/.
[4] We only ran on small to medium sized instances, due to time constraints.

- Best found solution cost (for the SAT problem) or %relative error (for the MAX-SAT and QAP problems) per run,
- Average fraction of best found solutions due to an aspiration move,
- Average number of better-than-previous solutions found per run,
- Average cost (for the SAT problem) or %relative error (for the MAX-SAT and QAP problems) of all better-than-previous solutions found per run,
- Average number of aspiration moves made per run.

These search monitors were used as they were the ones which gave different values when aspiration moves were added to GLS and thus best showed the effect that aspiration moves had on the search.

3.4 Results

In this section we show graphs of the results obtained from the experiments described in the previous section. For each relevant monitor, we plot the average values for all problems and all runs for each problem type. The x-axis represents the value of λ used (unless otherwise stated), and is logarithmic. In addition to this, many of the graphs have two lines plotted on them, one representing the results for GLS with aspiration and one for GLS without aspiration. So, for example, in Figure 3-2, the average cost of the best found solution, found by GLSSAT at $\lambda = 2$, is 0.94, and for GLSSAT with aspiration, it is 0.78. For full results tables, see the appendix.

3.4.1 Comparing GLS with and without aspiration

3.4.1.1 SAT Results

In Figure 3-2, the average cost (the number of unsatisfied clauses for SAT) of final best found solutions from each run of GLSSAT with and without aspiration are compared, for each value of λ in the set of values given earlier. The purpose of this

graph is to illustrate any performance differences between GLSSAT and GLSSAT with aspiration, over different parameter settings (i.e. different λ settings). It should be noted that lower values mean the algorithm performed better. For example, when λ is set to 1, GLSSAT with aspiration (denoted by the diamond shaped points) over all problems and all runs produces a final best found solution with on average 0.7 unsatisfied clauses and GLSSAT without aspiration moves (denoted by the square shaped points) having on average 0.82 unsatisfied clauses. Obviously, in this case GLSSAT with aspiration performed slightly better.

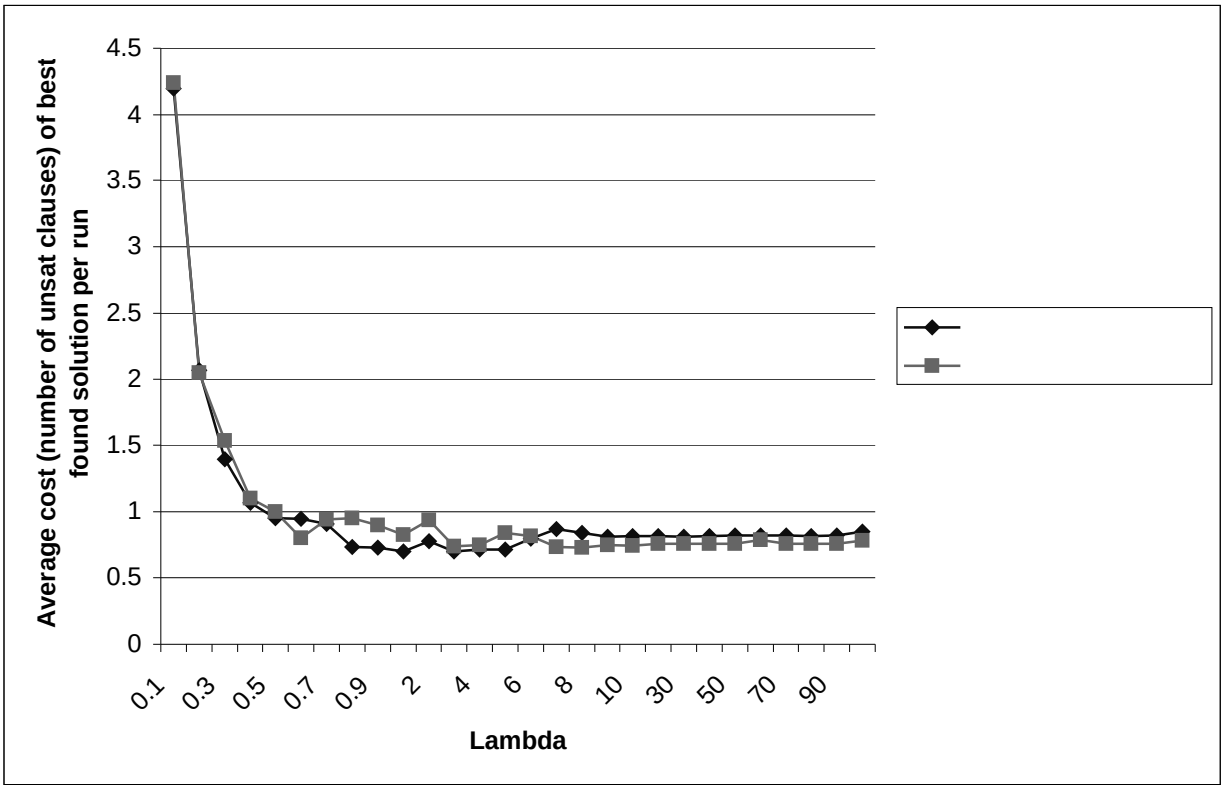


Figure 3-2: Average of number of unsatisfied clauses in final best found solution, for GLSSAT with and without aspiration in the SAT problem

From Figure 3-2 above, we can see that the average number of unsatisfied clauses (i.e. the cost) over all problems and all runs is sometimes slightly lower with aspiration (than without) between λ = 0.1 and λ = 7. From λ = 8 to λ = 100, GLSSAT

with aspiration gives marginally worse results than GLSSAT without. Using a sign test we have found that there is no significance difference between the pair of algorithms (see the Appendix for details of this). The sign test was conducted by taking the difference between the average number of unsatisfied clauses for each value of lambda for both algorithms, and then counting the number of positive and negative differences (if there is no difference between the values for a parameter setting, then that setting is ignored by the test). The probability of this number of positive and negative differences occurring by chance is then calculated using a binomial distribution. If the probability of this occurring is less than or equal to 5%, then the result is said to be statistically significant, otherwise it is said that there is no significant difference between the results (the interested reader should refer to a statistics book such as [13] for details of how to perform the sign test).

In Figure 3-3 below, we show the average fraction of best found solutions (see Definition 3.1) at the end of a a run found by an aspiration move.

$$\text{Fraction of best found solutions due to an aspiration move} = \frac{\#\text{best found solutions due to an aspiration move}}{\#\text{best found solutions}}$$

Definition 3.1 : Average fraction of best found solutions due to an aspiration move

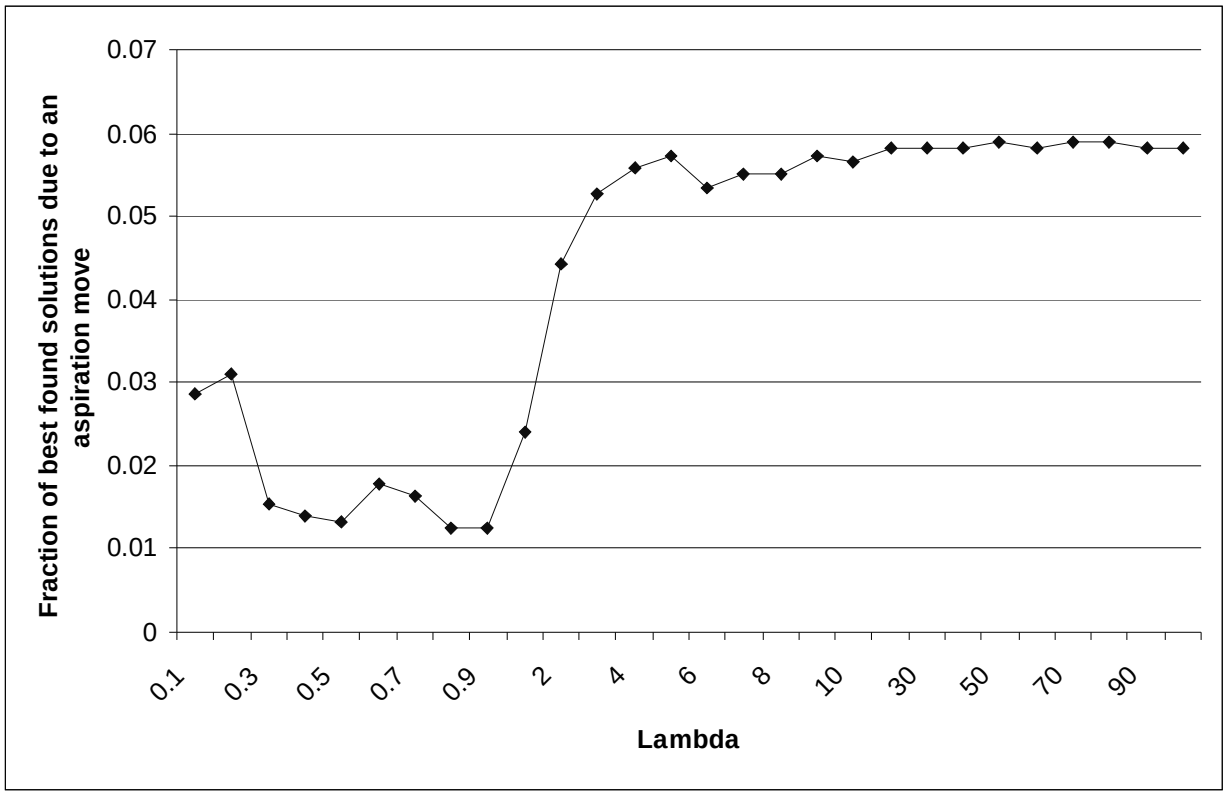


Figure 3-3: Average fraction of final best found solutions due to aspiration for GLSSAT in the SAT problem

From Figure 3-3, we can see that the fraction of best solutions found by GLSSAT with aspiration, due directly to aspiration, increases sharply from $\lambda = 1$ to $\lambda = 5$, (rising from about 0.012 to 0.055) and then stays roughly constant for λ values from 5 to 100 (at about 0.058). The sharp increase between $\lambda = 0.9$ and $\lambda = 3$ is different from the QAP case and also the MAX-SAT case (see the comparison and discussion sections).

In Figure 3-4, we measure the average number of *better-than-previous solutions* found during a run of GLSSAT compared to GLSSAT with aspiration, plotting a point for each value of λ tried. A better-than-previous solution is a solution whose cost is lower than all the previous solutions before it. Thus the set of all better-than-

previous solutions for a particular run of GLSSAT is the set of all solutions whose cost was better than all previous solutions, when it was found during the current run.

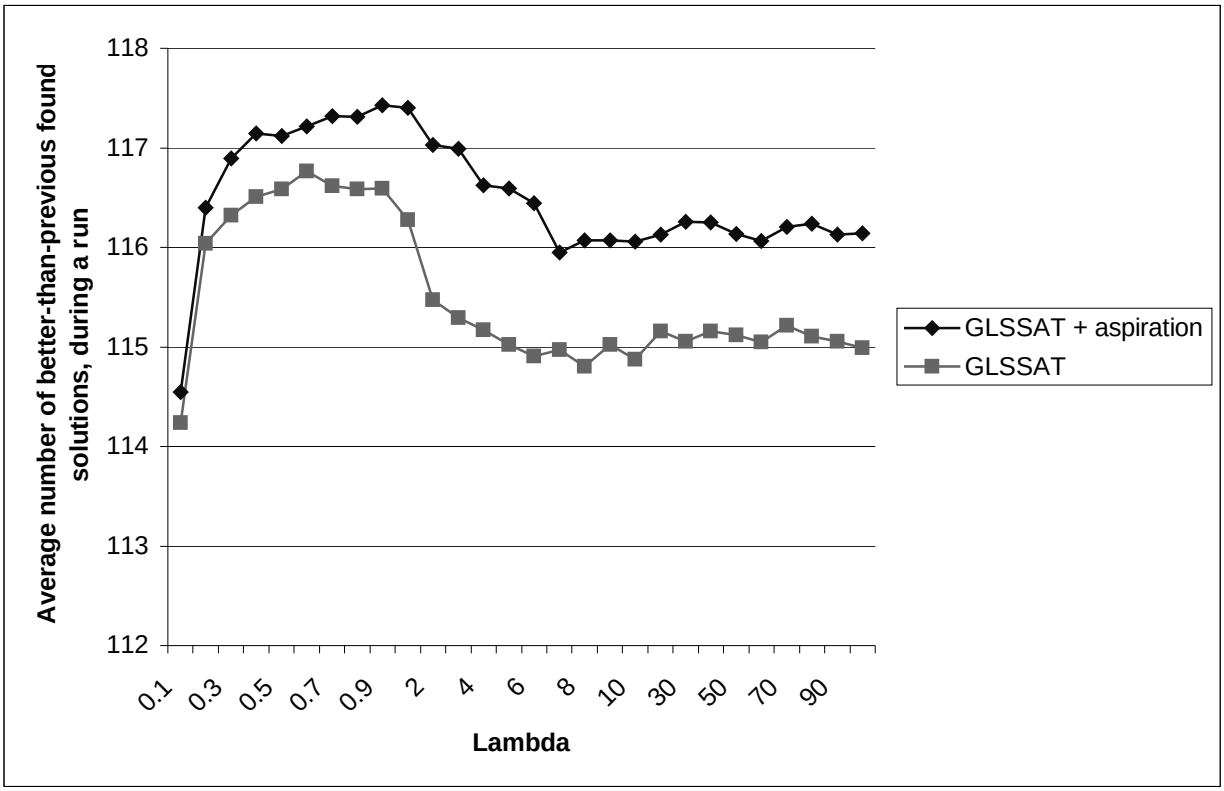


Figure 3-4: Average number of better-than-previous solutions found by GLSSAT with and without aspiration in the SAT problem

In Figure 3-4 above, we can see that the average number of better-than-previous solutions found over 10 runs on all problems, is higher with aspiration than without aspiration, with the gap tending to widen slightly as λ increases.

Figure 3-5 shows the average cost of better-than-previous solutions, over all problems and all runs of GLSSAT with and without aspiration. This is intended to illustrate the difference in the quality of better-than-previous solutions.

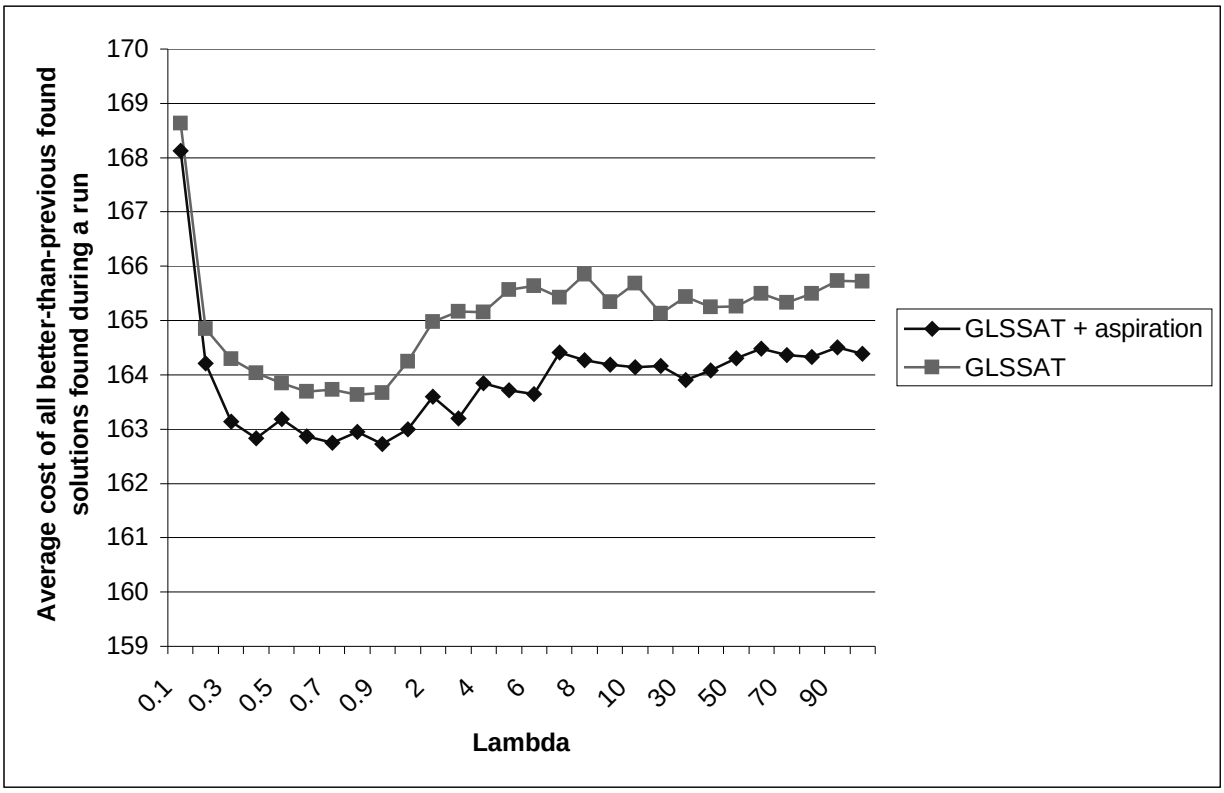
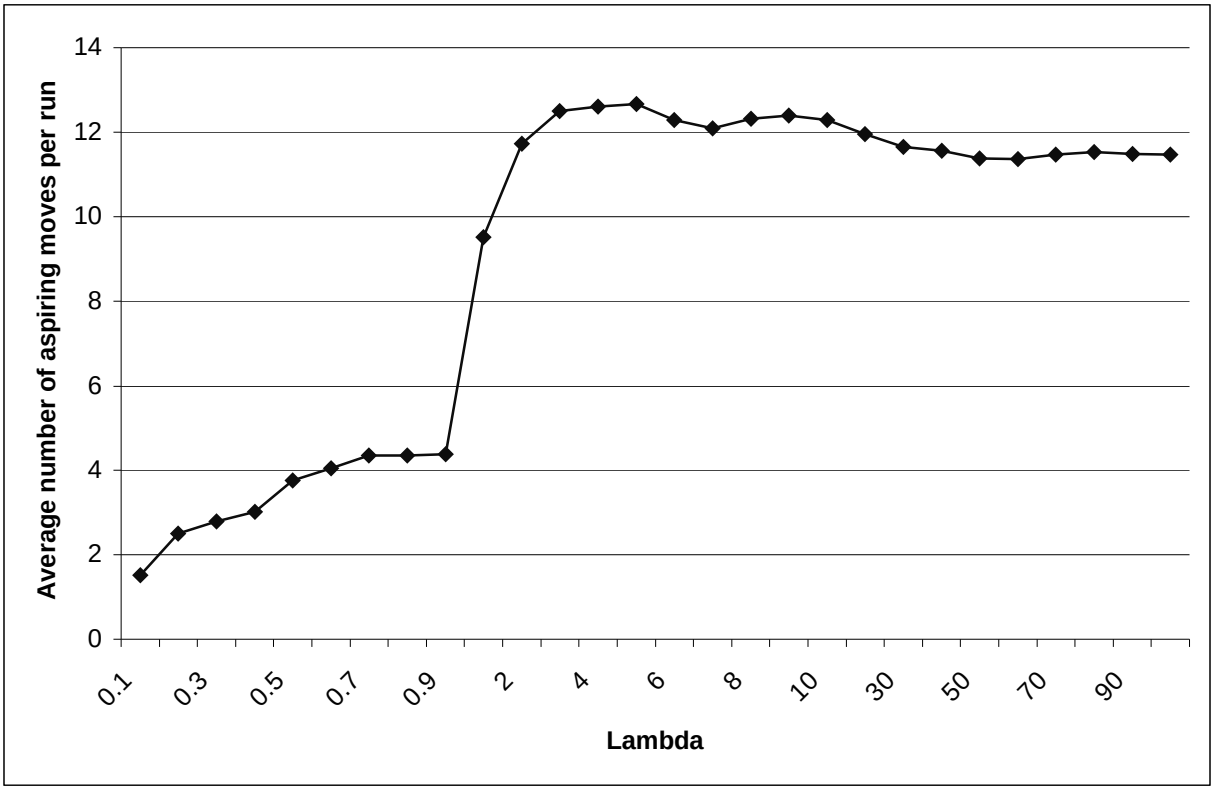


Figure 3-5: Average cost of all better-than-previous solutions found by GLSSAT with and without aspiration in the SAT problem

In Figure 3-5 above, we can see that GLSSAT with aspiration finds (on average) better-than-previous solutions with lower cost, than GLSSAT without aspiration. The gap again widens as λ increases.


Average number of aspiring moves per run
14
12
10
8
6
4
2
0
0.1
0.3
0.5
0.7
0.9
2
4
6
8
10
30
50
70
90
Lambda

Definition 3.2]) found by GLSMAXSAT and GLSMAXSAT with aspiration. This is intended to show the difference in performance of GLSMAXSAT with and without aspiration.

$$\%relative\_error(Cost) = \frac{Cost - Best\_known\_cost}{Best\_known\_cost} * 100$$

Definition 3.2: %relative error of a solution

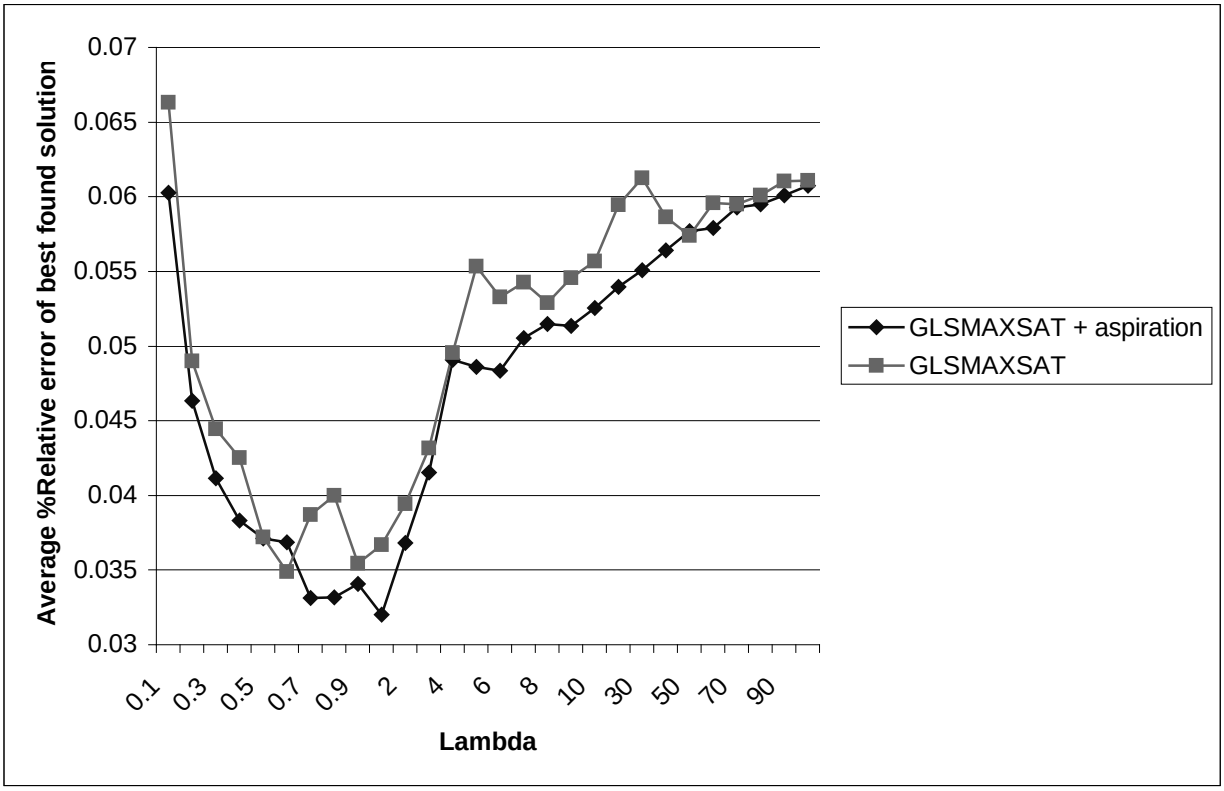


Figure 3-7: Average %relative error of the best found solution per run for GLSMAXSAT with and without aspiration in the MAX-SAT problem

Figure 3-7 shows that GLSMAXSAT with aspiration sometimes produces solutions of better quality than GLSMAXSAT without aspiration, particularly for values of λ between 5 and 30. We have applied a sign test to the two series of data and have found that based on this, there is evidence that GLSMAXSAT with aspiration moves performs better than GLSMAXSAT without aspiration moves (see Appendix for details of this).

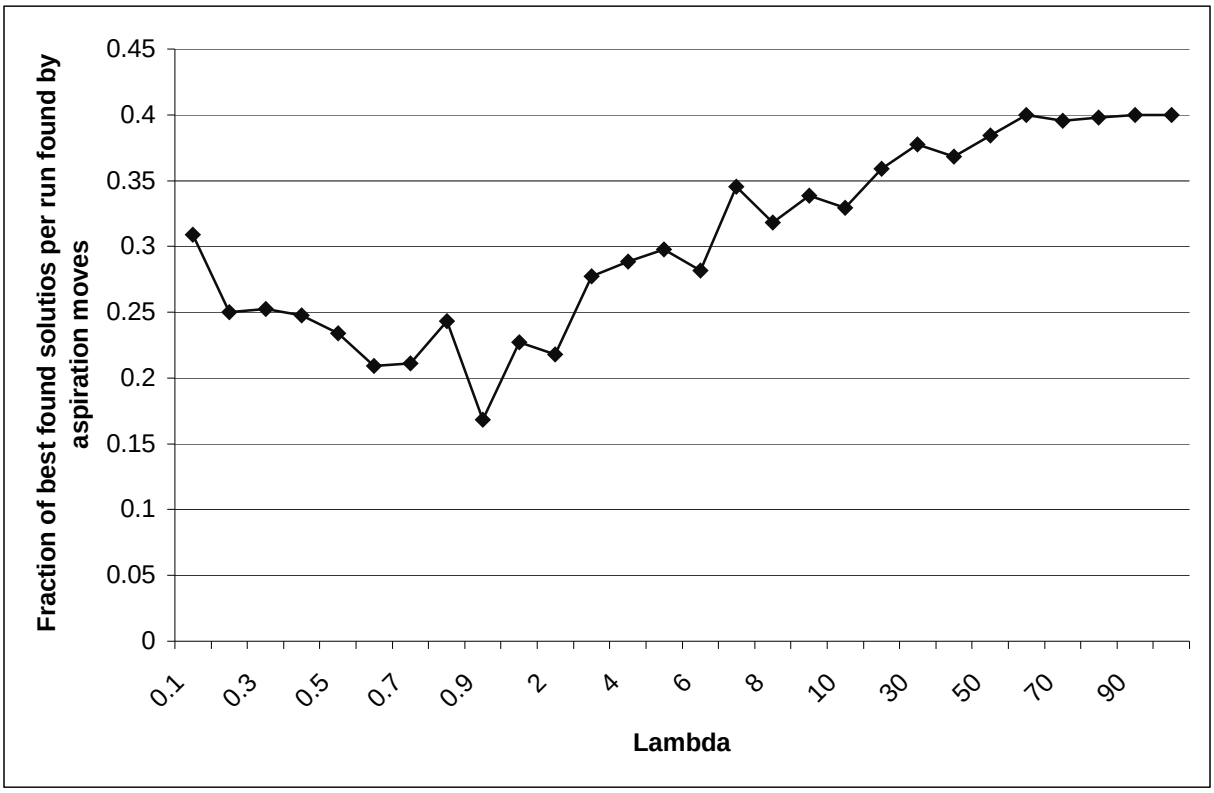


Figure 3-8: Fraction of best found solutions, found by an aspiration move for GLSMAXSAT in the MAX-SAT problem

Figure 3-8 above shows that the fraction of best found solutions per run, due directly to an aspiration move, increases slightly (from about 0.17 to 0.4) as λ increases from 1 to 100.

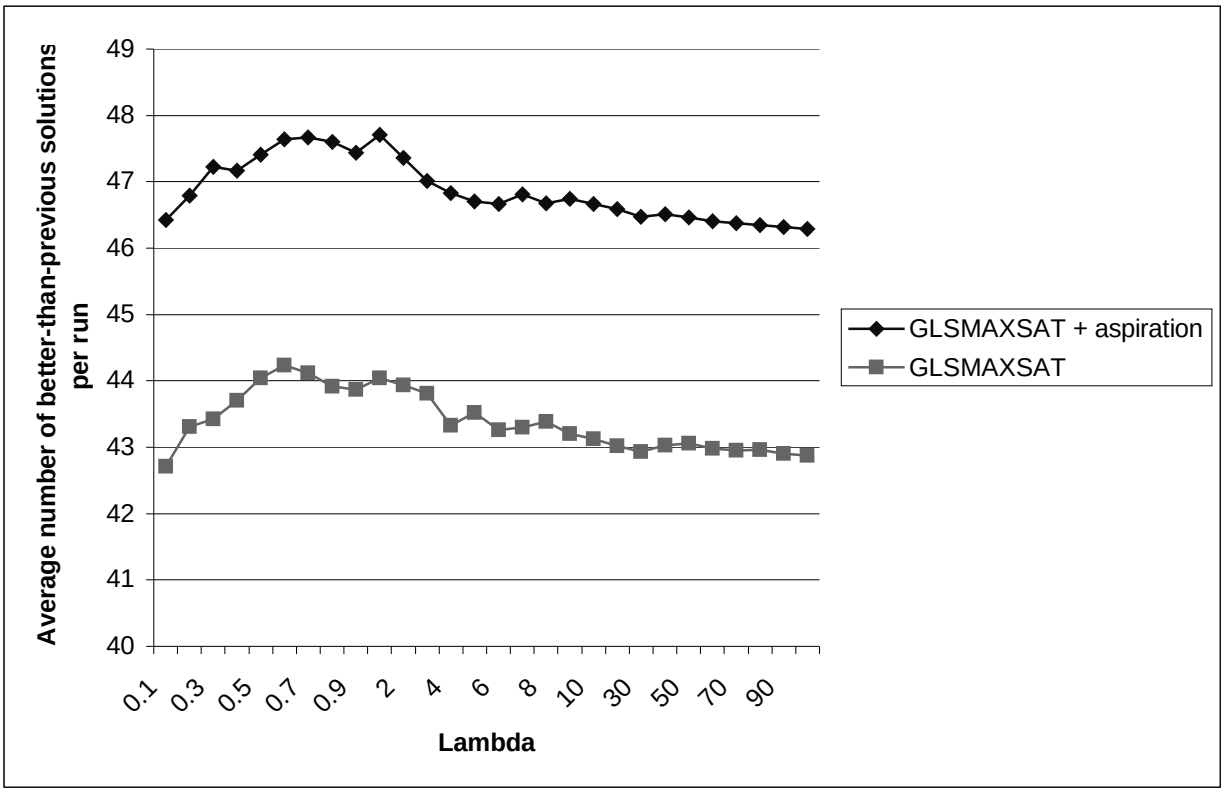


Figure 3-9: Average number of better-than-previous solutions per run for GLSMAXSAT with and without aspiration in the MAX-SAT problem

Figure 3-9 shows that slightly more (about 3 per run) better-than-previous solutions (see definition in previous section) are found on average by GLSMAXSAT with aspiration than without.

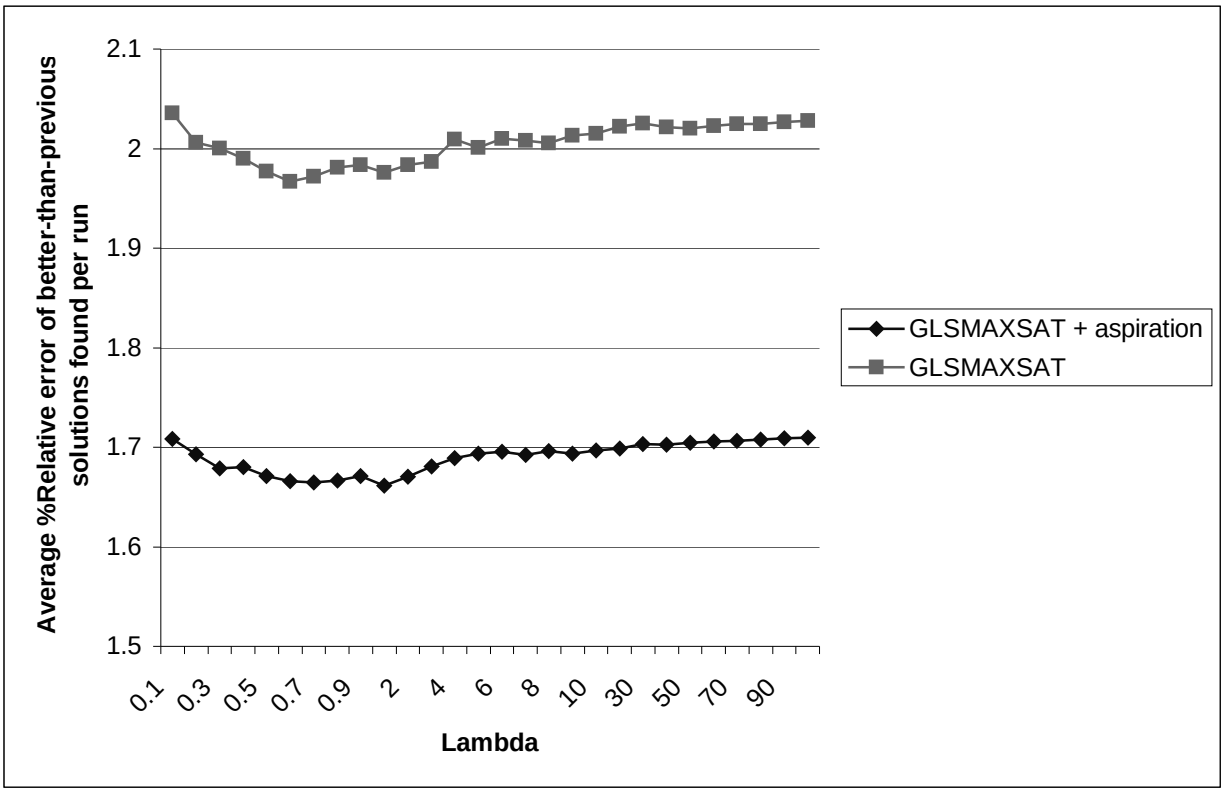


Figure 3-10: Average %relative error of all better-than-previous solutions per run, over 10 runs in the MAX-SAT problem

Figure 3-10 shows that the average %relative error of all better-than-previous solutions per run is lower (so that these solutions are of better quality) for GLSMAXSAT with aspiration moves than without.

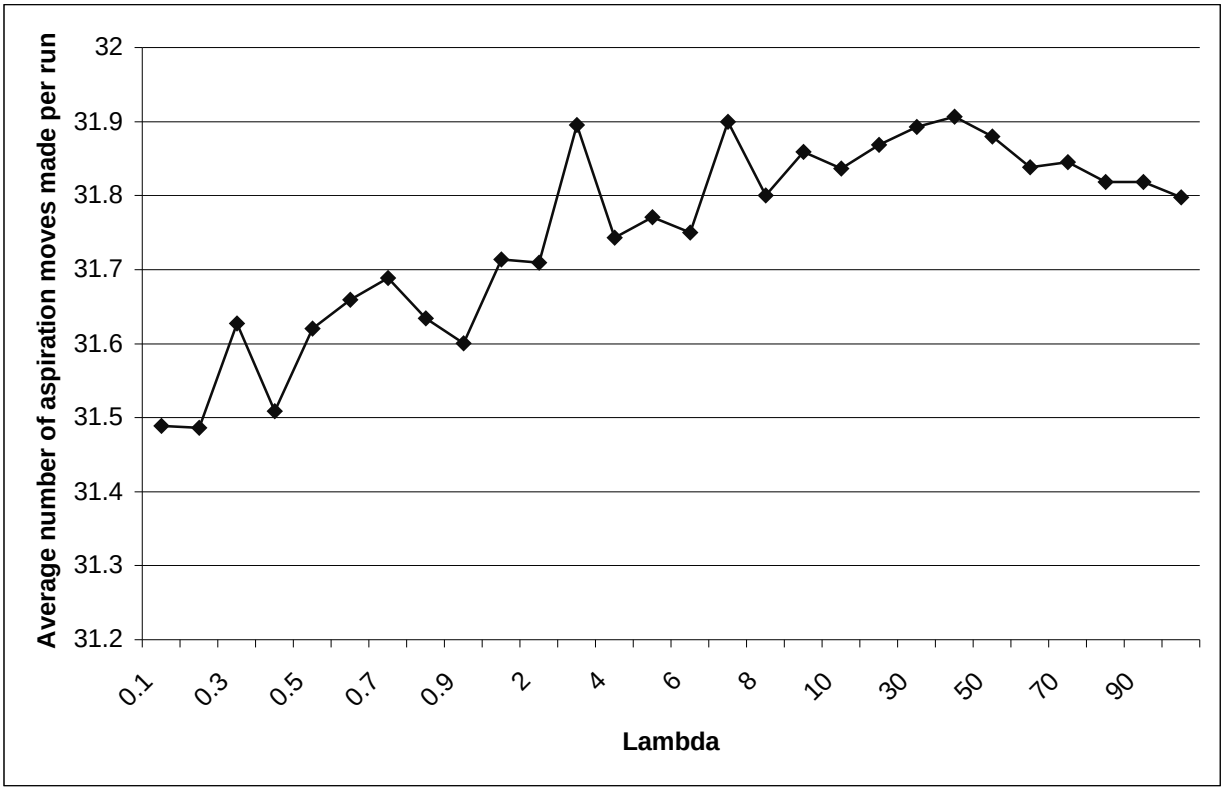
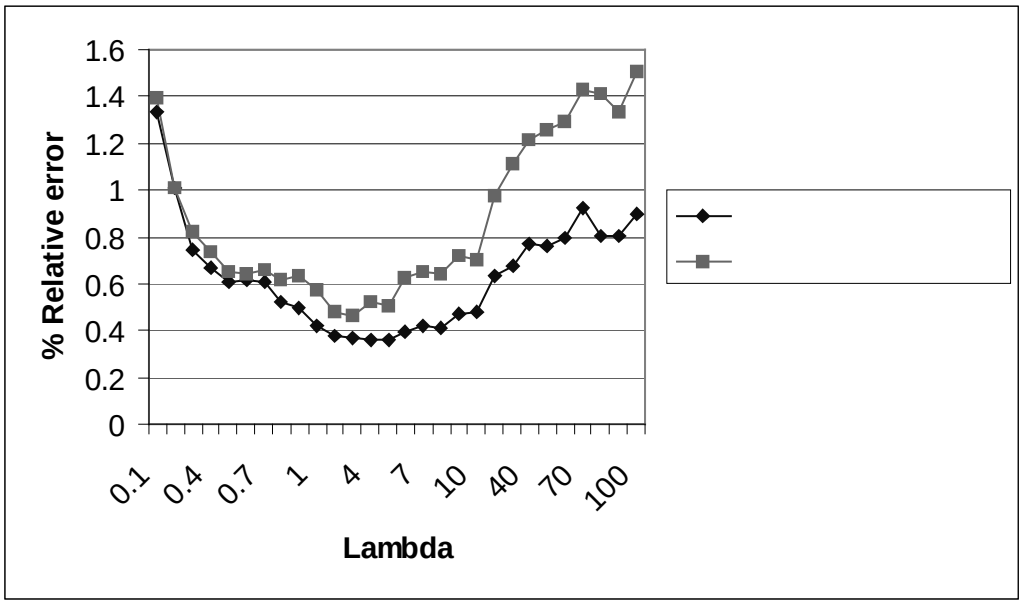


Figure 3-11: Average number of aspiration moves made per run for GLSMAXSAT with aspiration in the MAX-SAT problem

Figure 3-11 above shows that the number of aspiration moves made with GLSMAXSAT generally increases (although very slowly) with increasing values of λ.

3.4.1.3 QAP Results

The graph below gives the %relative error (see Definition 3.2) of the average best found solution from the best known solution cost for both GLSQAP and GLSQAP with aspiration.


1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
% Relative error
0.1
0.4
0.7
1
4
7
10
40
70
100
Lambda

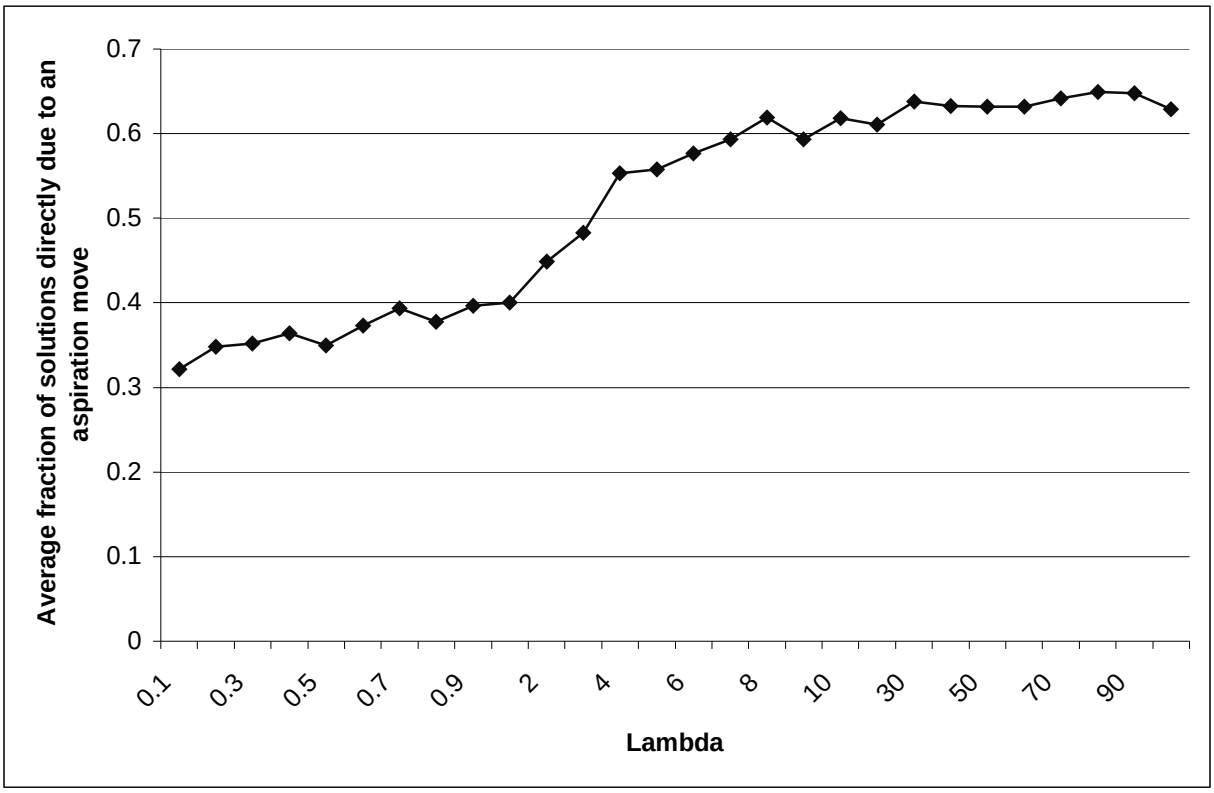
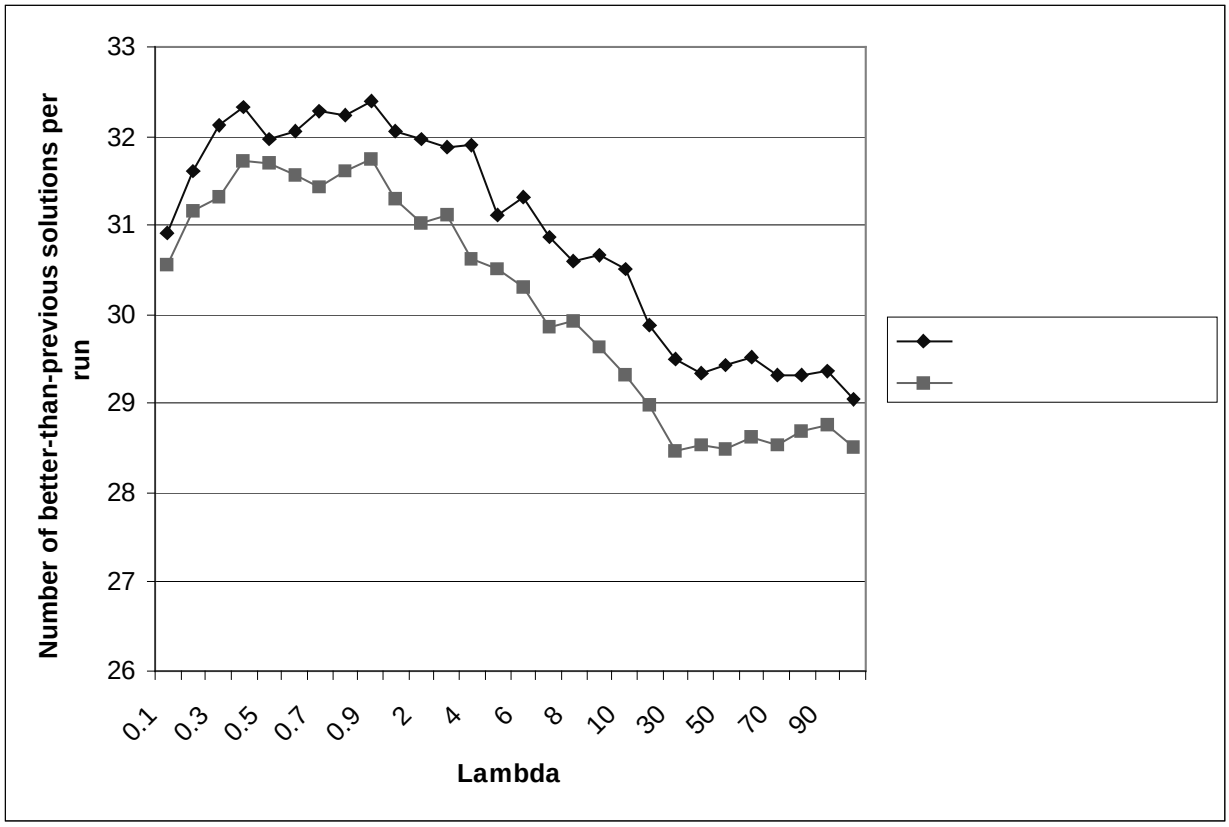
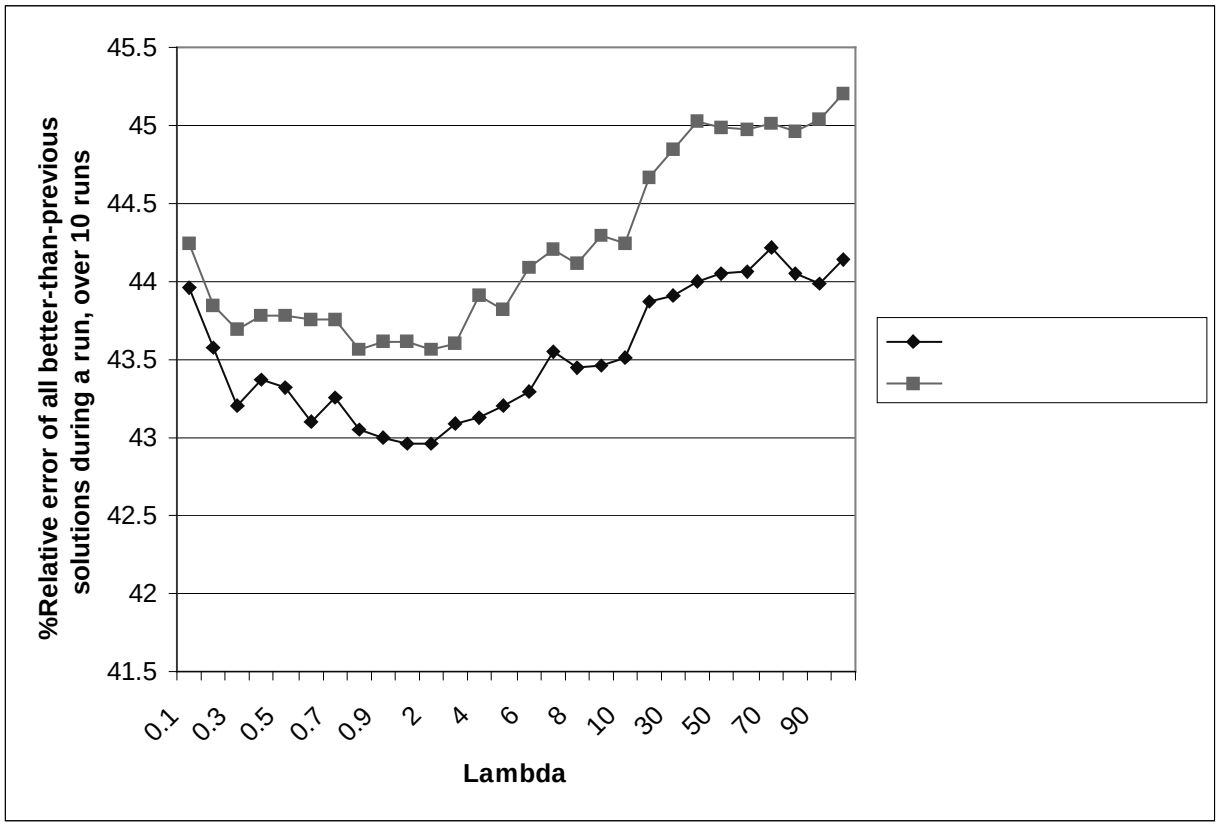
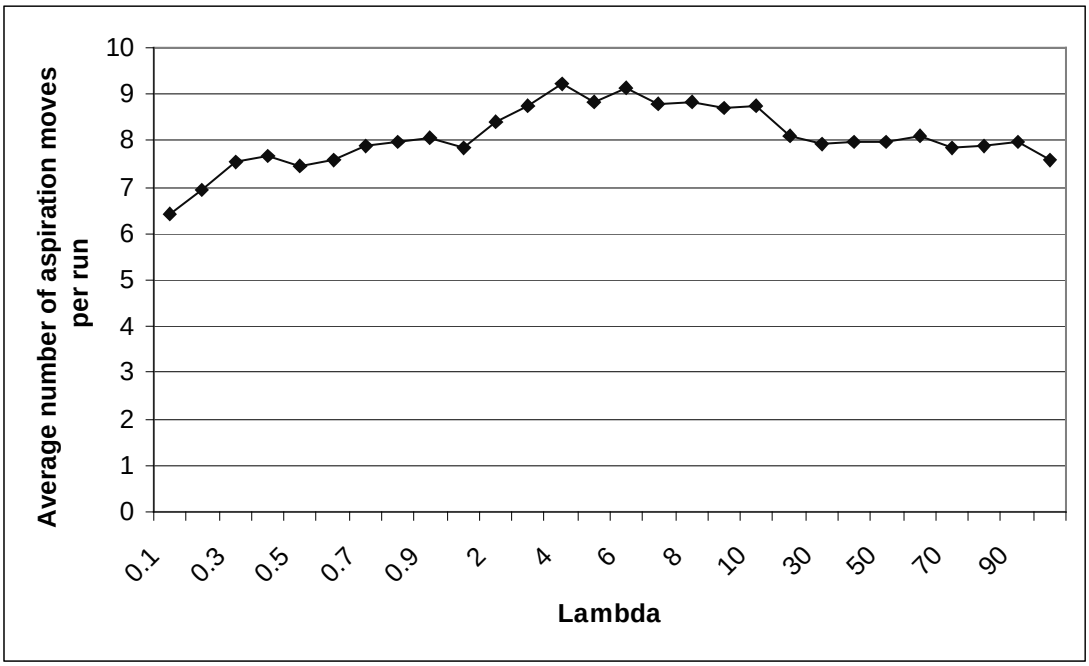


Figure 3-13: Average fraction of best found solutions found by an aspiring move, for GLSQAP with aspiration in the QAP

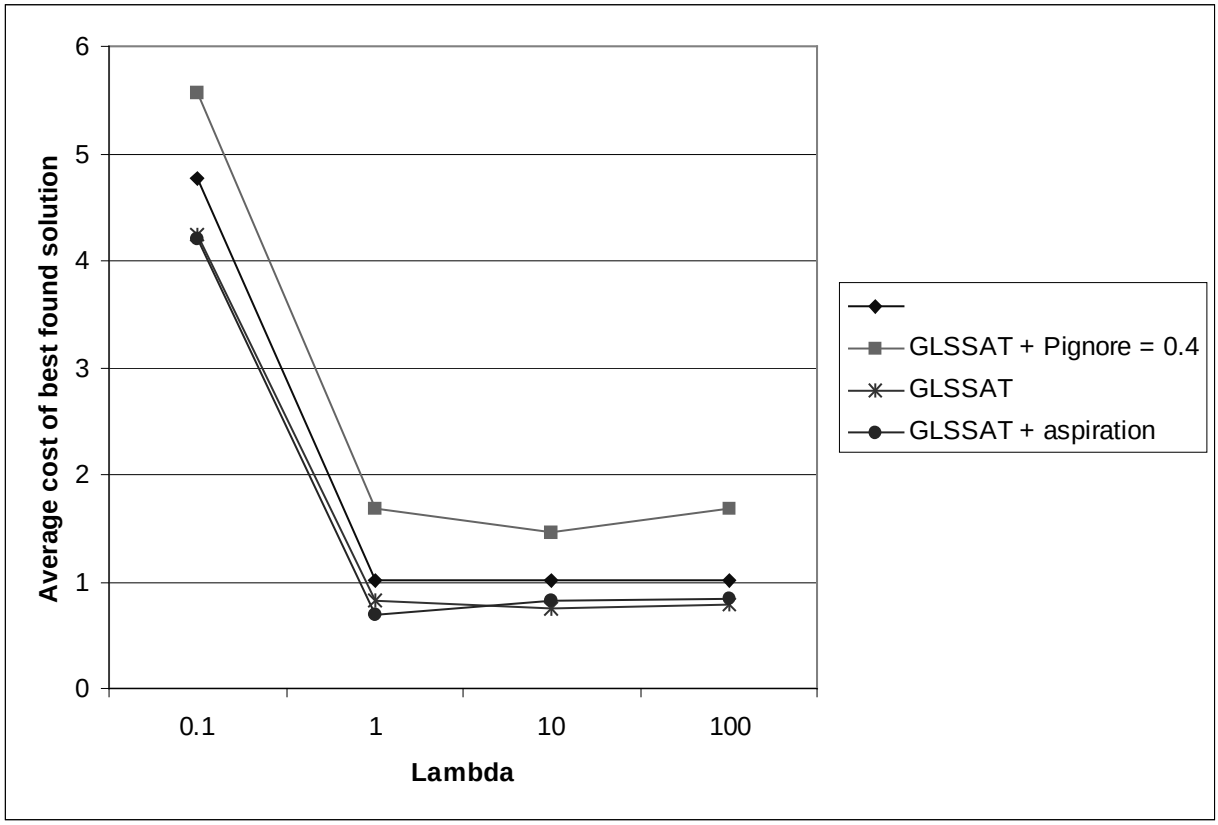
In Figure 3-13 above we can see that, as the λ coefficient increases from 1 to 10, the fraction of best found solutions (see Definition 3.1) directly due to aspiration increases from 0.3 to about 0.64 (when the λ coefficient is in the range 10-100).


Number of better-than-previous solutions per run
33
32
31
30
29
28
27
26
0.1
0.3
0.5
0.7
0.9
2
4
6
8
10
30
50
70
90
Lambda


%Relative error of all better-than-previous solutions during a run, over 10 runs
45.5
45
44.5
44
43.5
43
42.5
42
41.5
0.1
0.3
0.5
0.7
0.9
2
4
6
8
10
30
50
70
90
Lambda


Average number of aspiration moves per run
10
9
8
7
6
5
4
3
2
1
0
0.1
0.3
0.5
0.7
0.9
2
4
6
8
10
30
50
70
90
Lambda

3.4.2.1 SAT Results



3.4.2.2 MAX-SAT Results

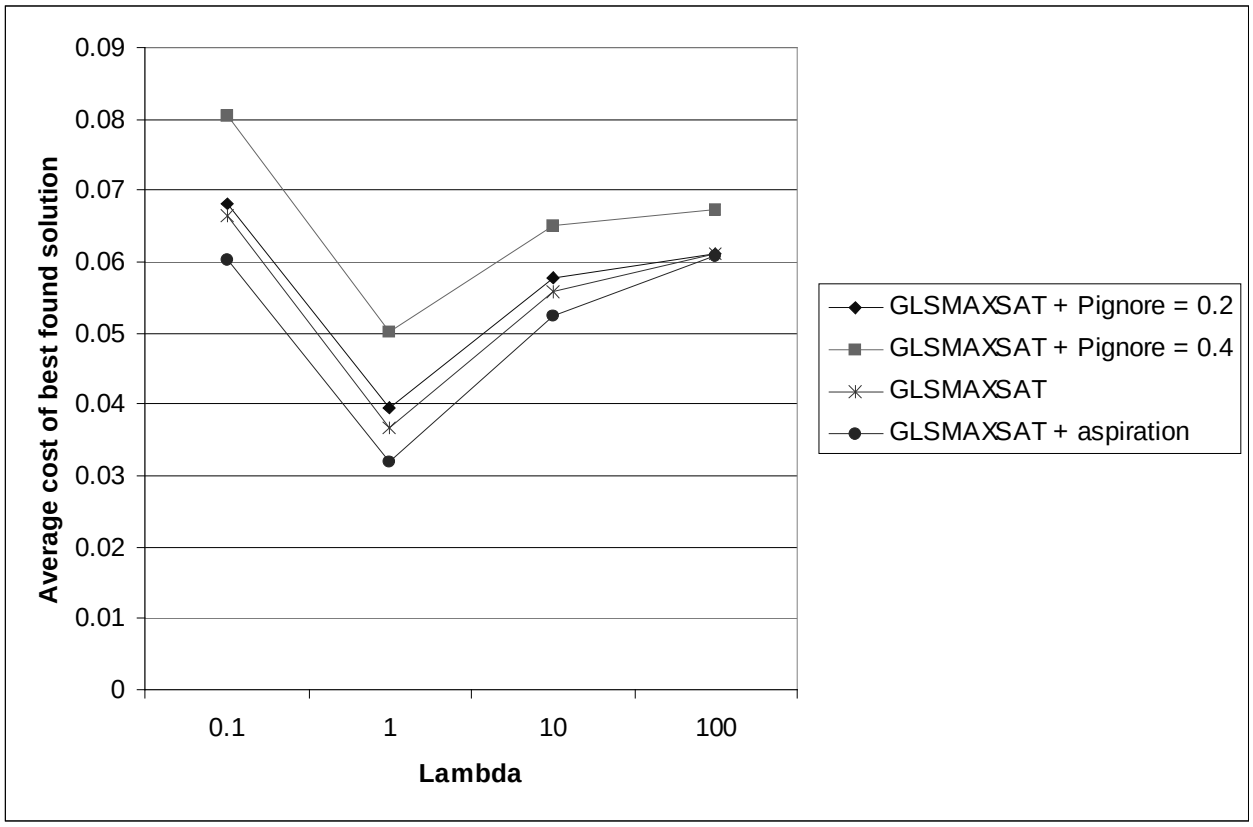


Figure 3-18: GLSMAXSAT and GLSMAXSAT + aspiration versus GLSMAXSAT with probabilistically ignoring penalties in the MAX-SAT problem

Figure 3-18 shows that, for GLSMAXSAT, ignoring penalties does not produce better quality solutions over standard GLSMAXSAT (or GLSMAXSAT with aspiration) and, when higher probabilities are used, the solution quality becomes worse.

3.4.2.3 QAP Results

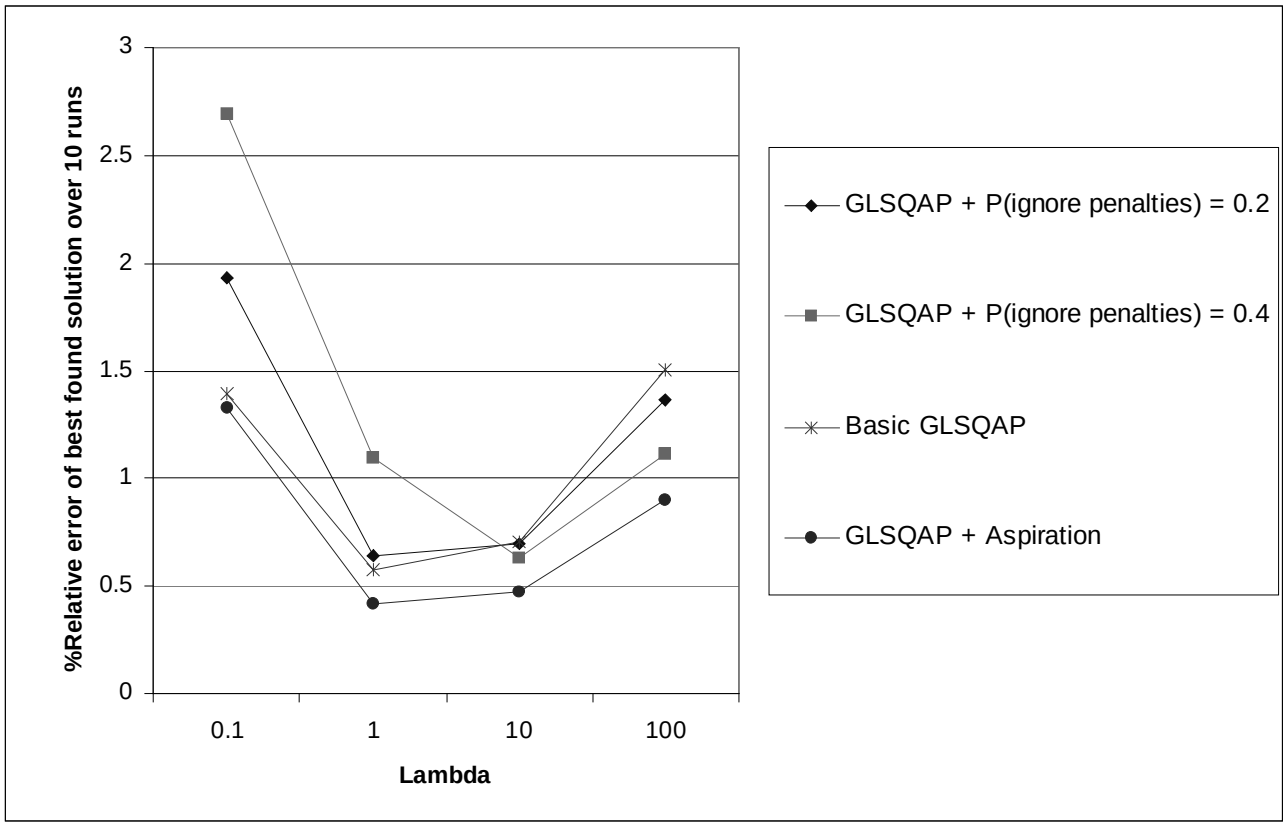


Figure 3-19: The effect of probabilistically ignoring the penalty term of the augmented objective function for GLSQAP, in comparison with basic GLSQAP and GLSQAP with aspiration in the QAP

Figure 3-19 shows that picking a move with a certain probability of ignoring the penalty term of the augmented objective function does not produce such good-quality solutions as GLSQAP with aspiration, although for P(ignore penalties) = 0.4, when λ = 100 there is a slight improvement over the basic GLSQAP scheme, but much worse results are found (than for GLSQAP with aspiration) when the λ coefficient is set to 0.l and 10. When higher values of P(ignore penalties) are used (0.6 and 0.8), worse results are obtained for all values of λ.

3.5 Comparison

From the results in the previous section, it is clear that the *best-improved aspiration criterion* can be useful to GLS, particularly in the Quadratic Assignment Problem, where variable cost features may lead to some features being penalised which, later in the search, may become irrelevant. In the SAT problem it is less clear that aspiration moves are useful, although in some cases, clearly better results were obtained and GLSSAT with aspiration moves does not perform worse than basic GLSSAT. In the weighted MAX-SAT problem, GLSMAXSAT with aspiration performs better overall, although for a few settings of the λ parameter it performs slightly worse. Overall it therefore appears that the best-improved aspiration moves are a useful extension of GLS, particularly for optimisation problems and problems where the only suitable formulation of features for GLS is where the cost of features varies during a run of GLS (for example, the QAP).

The improved-best aspiration criterion allows GLS to visit more, better quality, better-than-previous found solutions, as it allows the local search algorithm to visit solutions which otherwise might have been blocked off due to penalties imposed earlier in the search, and thus which might have been missed. From our experiments with ignoring penalties, it is clear that the reason that the best-improved aspiration criterion works is not simply because every so often it allows GLS to ignore the penalties, since simply ignoring penalties with a certain probability does not produce the increased solution quality that adding the aspiration criterion does. The reason that the best-improved aspiration criterion works is because *when a potential new best found solution is present in the neighbourhood* of the local search algorithm, it allows

the local search algorithm to visit this new best solution *regardless of the penalties already imposed on features of that new best solution*.

3.6 Discussion

Aspiration moves are less useful for problems like the SAT problem, with long plateaus leading to lower long plateaus. They work better for problems like the QAP, which contain many local minima basins. However, why they are more useful for problems with more local minima is a topic for further investigation and thought.

As well as all the results reported in the results section, we also looked at all the other search monitors apart from those involving either the final best found solution or those involving better-than-previous solutions. We found that these did not vary much with or without aspiration. For example, aspiration moves did not affect the average cost of local minima or the average cost of all solutions visited during a run. This was probably because aspiration moves do not change the course of the search very much, and only account for a very small fraction of moves made (e.g. approximately 2-12 out of 10*n for SAT, approximately 31 out of 10*n for MAX-SAT, approximately 6-9 moves per run out of 1000*n for QAP. See Figure 3-6, Figure 3-11 & Figure 3-16).

There are a number of things we wanted to attempt, but which will have to be left to future work, owing to time considerations. These included experimenting with the effect of the maximum number of repairs allowed for GLS with and without aspiration, and investigating whether this affected the results. We believe that, as the number of repairs is increased, the effect of the aspiration criterion will decrease as the basic GLS without aspiration criterion will have more opportunity to find better solutions.

Another aspect we would have liked to have experimented with is to try a more advanced aspiration criterion. This would allow the penalties to be ignored if a solution existed, such that it was of better quality (lower cost) than the worst of the best *Q* solutions visited so far. Aspiration moves to the current *Q* best found solutions would not be allowed, thus ensuring that this scheme will not cause solutions to be revisited. This would allow us to vary the number of aspiration moves. Thus, we would be able to see if allowing more aspiration moves had any effect on the quality of solutions and if it would be possible to produce even better results. Hopefully, this would lead to GLS exploring higher quality basins and plateaus in the search landscape, which might otherwise have been ignored owing to penalties imposed earlier on in the search.

3.7 Conclusion

In this chapter, we have shown how GLS may be extended to include the best-improved aspiration criterion previously used with tabu search algorithms. We have shown that this generally does not make the performance of GLS worse, and in many cases improves the performance of GLS, in terms of solution quality. We have also performed experiments to confirm our intuition as to why aspiration moves work with GLS, and have produced some evidence to support these hypotheses, namely that the best-improved aspiration criterion works partly due to the condition under which an aspiration move is applicable (i.e. when there exists a move that can generate a new best found solution, regardless of the penalties imposed on that solution). This helps to prevent GLS from missing these important solutions, because of penalties imposed earlier in the search. This is particularly useful for GLSQAP, as it allows higher values of λ to be used, without such a large degradation of performance (as without

aspiration moves), thus increasing the range of λ settings under which GLS can work acceptably well. Since aspiration moves do not usually decrease solution quality and in many cases improve solution quality, we believe that they are an extension of GLS that should become a standard part of the GLS.

4 Random moves

In this chapter we give our motivation for random moves (or some other mechanism to create the same effect) to be added to Guided Local Search and show how this may be done. We then present experimental results on three groups of problems: the SAT problem, the QAP problem, and the weighted MAX-SAT problem. Finally, we put forward some theories as to when and why random moves may improve the performance of Guided Local Search, based on our experimental results.

4.1 Motivation

Guided Local Search works very well, as long as a good value for the λ parameter is used. If the value is too high, GLS performs less well, because the search becomes too diverse, preventing the algorithm from getting deep into local minima and thus finding good solutions. If the value is too low, GLS also performs worse, because the search becomes too intense, searching plateaus and local minima in too much detail, so that it never gets to search new areas of the search space. Thus, the motivation behind random moves is to try to prevent GLS getting "stuck" in one part of the search space (for example, when λ is too low) and force it to move into other areas of the search space that it might not otherwise visit.

4.2 Adding random moves to GLS

Random moves can be added to GLS, by simply augmenting the local search algorithm, so that with probability $1\text{-}P_{randmove}$ we allow the local search algorithm to pick the best move from the neighbourhood and with probability $P_{randmove}$ we pick a move at random from the neighbourhood (with equal chance of picking any move). This random move extension of GLS was inspired by the work by Selman et al.

[79,59], who use a similar scheme in conjunction with their GSAT and Walksat algorithms (which are specific to the SAT problem). In addition to this, we also tried two other similar schemes specifically for the SAT and MAX-SAT problems. With probability $P_{randwalk}$, we select an unsatisfied clause at random (with all unsatisfied clauses having equal chance of being picked), and then pick a variable at random in that clause (all variables in that clause having equal chance of being picked) and flip it (otherwise, with probability $1\text{-}P_{randwalk}$ the standard local search scheme would be followed); this is the same as the random walk component in the walksat algorithm [79]. In the second scheme, which we call random penalty walk (an extension of the previous one), with probability $P_{randpenaltywalk}$, we select an unsatisfied clause from the set of all unsatisfied clauses at random, with the probability of picking any clause weighted according to $P_{pick\_clause\_randpenaltywalk}$ (see Definition 4.1 below). Then, a variable chosen at random from the chosen clause (with all the variables in that clause having equal chance of being chosen) would be flipped.

$$P_{pick\_clause\_randpenaltywalk}(C_j) = \frac{1 + \lambda \cdot a \cdot p_{cj}}{\sum_{C_i \in UnsatisfiedClauses} (1 + \lambda \cdot a \cdot p_{ci})}$$

where: λ = GLS search intensity parameter

a = GLS problem specific penalty weight (1 for SAT)

p_{ci} = the penalty associated with clause C_i

UnsatisfiedClauses = the set of all clauses unsatisfied in the current solution

Definition 4.1: Probability of picking an unsatisfied clause, for random penalty walk

```
Local_Search_Random_Move(x,h,g,N)
{
   do
   {
     if (withprobability(P_randmove))
     {
        y = solution picked at random from N(x)
        x = y
        Δh = h(y) - h(x)
        randmove = true
     }
     else
     {
        y = solution in N(x) such that h(x) is minimised,
            breaking ties randomly
        Δh = h(y) - h(x)
        if (Δh <= 0) x = y
        randmove = false
     }

     if (Δh = 0) sideways = sideways + 1
     else        sideways = 0

     if (g(x) < g(x*)) x* = x
   }
   while ((Δh <= 0) or randmove) and (sideways < 2)

   return x
}


//NOTE: SAT and MAX-SAT only
Local_Search_Random_Walk(x,h,g,N)
{
   do
   {
      if (withprobability(P_randwalk))
      {
        c = clause picked at random from the set of all
            unsatisfied clauses
        y = solution picked at random from the set of all solutions
            resulting from flipping one of the variables in c
        x = y
        Δh = h(y) - h(x)
        randwalkmove = true
      }
      else
      {
        y = solution in N(x) such that h(x) is minimised,
            breaking ties randomly
        Δh = h(y) - h(x)
        if (Δh <= 0) x = y
        randwalkmove = false
      }

      if (Δh = 0) sideways = sideways + 1
      else        sideways = 0

      if (g(x) < g(x*)) x* = x
   }
   while ((Δh <= 0) or randwalkmove) and (sideways < 2)

   return x
}
```

```
//Note: SAT and MAX-SAT only
Local_Search_Random_Penalty_Walk(x,h,g,N)
{
   do
   {
      if (withprobability(P_randpenaltywalk))
      {
         c = clause picked randomly weighted according to
             Definition 4.1 from the set of all unsatisfied clauses
         y = solution picked at random from set of all solutions
             resulting from flipping one of the variables in c
         x = y
         Δh = h(y) - h(x)
         randpenaltywalkmove = true
      }
      else
      {
         y = solution in N(x) such that h(x) is minimised,
             breaking ties randomly
         Δh = h(y) - h(x)
         if (Δh <= 0) x = y
         randpenaltywalkmove = false
      }

      if (Δh = 0) sideways = sideways + 1
      else        sideways = 0

      if (g(x) < g(x*)) x* = x
   }
   while ((Δh <= 0) or randpenaltywalkmove) and (sideways < 2)

   return x
}
```

where:

- *g()* returns the cost of a solution with regard to the original cost function,
- *h()* returns the augmented cost of a solution,
- *x** is the solution of lowest (original) cost found so far by the algorithm,
- *N(x)* is the neighbourhood function, giving neighbouring solutions of x

Figure 4-1: Pseudo code for local search with random moves, random walk and random penalty walk

4.3 Experiments

For all the SAT and MAX-SAT experiments in this section, we allowed 10**n* repairs and 10 runs (*n* = the number of variables in the problem). For all the QAP experiments in this section, we allowed GLSQAP 1000**n* repairs and averaged the results of 10 runs. For the SAT and MAX-SAT problems, if a solution with no

unsatisfied clauses was found, we restarted the local search from a random start point. This was to ensure that the algorithm always ran for the same number of iterations, so that the search monitors would be consistent over different runs of GLS, regardless of whether a solution was found or not during the run.

4.3.1 Preliminary tuning of GLS with random moves

The experiments for random moves were performed in two phases. First, a set of experiments were run to find suitable values for the probabilities of $P_{randmove}$ (for all problems) and $P_{randwalk}$ and $P_{randpenaltywalk}$ (for the SAT and MAX-SAT problems only[7]). Obviously different probabilities may work better or worse for different values of λ. However, testing all combinations of λ and values for the probabilities would involve too many tests. Thus, we limited our first set of experiments to run over the set of λ values { 0.1, 1, 10, 100 }, varying the probabilities for the SAT and MAX-SAT over the set of values { 0.001, 0.002, 0.004, 0.006, 0.008, 0.01, 0.05, 0.1, 0.2, 0.4, 0.6, 0.8 }[8] and for the QAP over the set of values { 0.2, 0.4, 0.6, 0.8 }.

4.3.2 Extended evaluation of GLS with random moves

For the second phase, the approximately-tuned GLS algorithms were run over the set of λ values { 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 } to assess the effect that the random move extensions had on GLS for each problem type and when and why they had that effect. The approximately tuned parameter values used for each extension and problem are shown in Table 2 below.

[7] It should be noted that random walk and random penalty walk are only applicable to the SAT and MAX-SAT problems, as they are specifically designed for these problems and not the QAP.

[8] The reason why only the SAT and MAX-SAT were tested with the lower probabilities is, because when we used only the set {0.2,0.4,0.6,0.8} we could not find good values for the probabilities, so we then added the lower probabilities, to see if these would perform better.

Parameter	SAT	MAX-SAT	QAP
$P_{randmove}$	0.01	0.006	0.2
$P_{randwalk}$	0.1	0.006	NA
$P_{randpenaltywalk}$	0.2	0.004	NA

Table 2: Approximately tuned parameter values for GLS random move extensions

4.4 Results

In this section, we present results of the experiments outlined in the last section. For each graph the x-axis represents the value of the GLS λ parameter, and the y-axis represents the average value for the search monitor over all runs and all problems tried for each variant of GLS which was tested. The legend shows which line (or bar of the bar chart) of the graph corresponds to which GLS variant. For example, in Figure 4-2, the third bar of each set in the bar chart corresponds to GLSSAT with $P_{randmove}$ set to 0.004, so for example when λ is 10, this variant of GLSSAT has an average best found solution cost of 0.64 over 10 runs for all problems it was tested on. A typical example of the graph obtained is Figure 4-5, where each line represents a different variant of GLSSAT, the differently shaped points on each line corresponding to different variants of GLSSAT. In this case, the basic GLSSAT is represented by square points, GLSSAT with random moves by diamond points, GLSSAT with random walk by triangular points, and GLSSAT with random penalty walk by crosses. So, for example, the average best-found solution cost, when λ is 0.8, for GLSSAT with random moves is 0.83. For the SAT results, we have used the number of unsatisfied clauses (i.e. solution cost) rather than %relative error, since this is the usual way in which the quality of solutions to problems in SAT are judged if no solution was found which satisfies all the clauses.

4.4.1 SAT Results

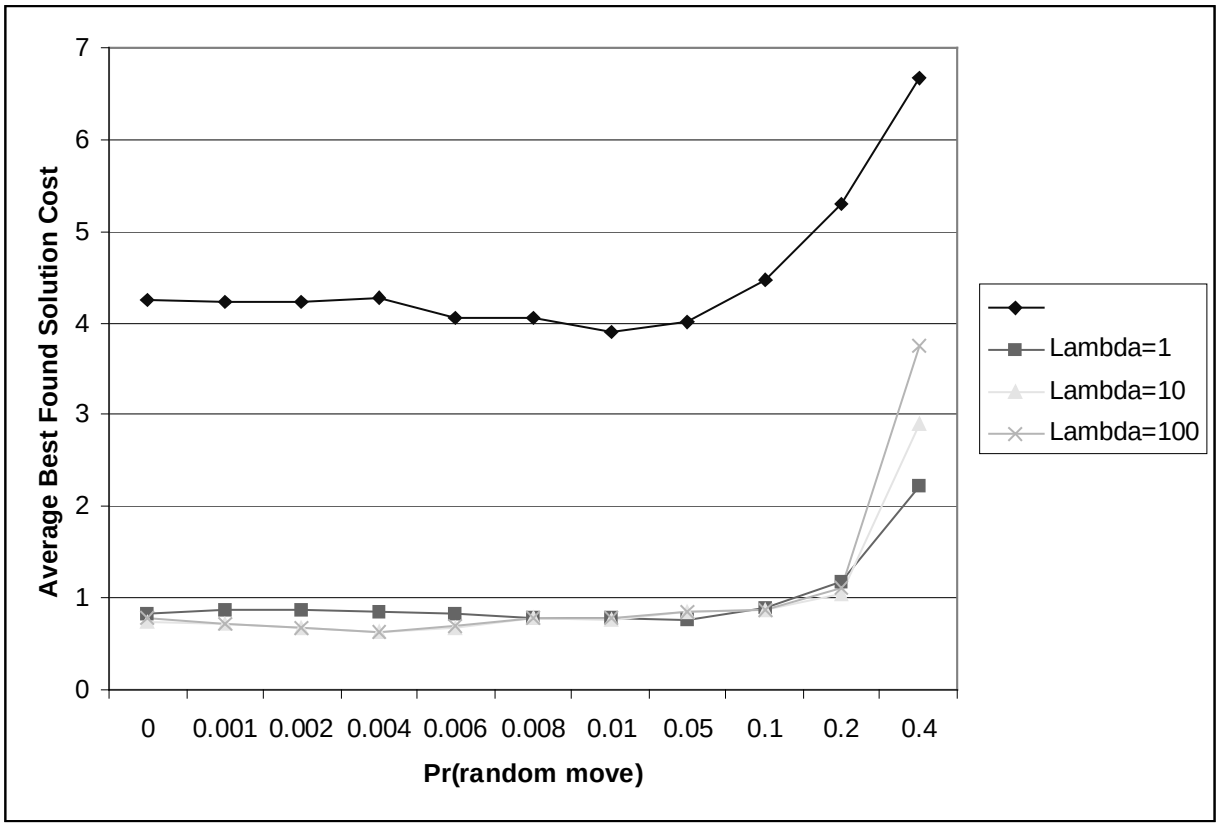


Figure 4-2: Results of the tuning phase for GLSSAT with random moves

Figure 4-2 shows the variation in performance of GLSSAT with different values of λ and different values of $P_{randmove}$, the probability of executing a move picked at random from the local search neighbourhood. This shows that a probability of making a random move of 0.1 or more produces a detrimental effect on the average performance of GLSSAT. In fact, even with lower probabilities of making a random move, little performance appears to be gained, with the best performing probability for picking a random move appearing to be 0.01.

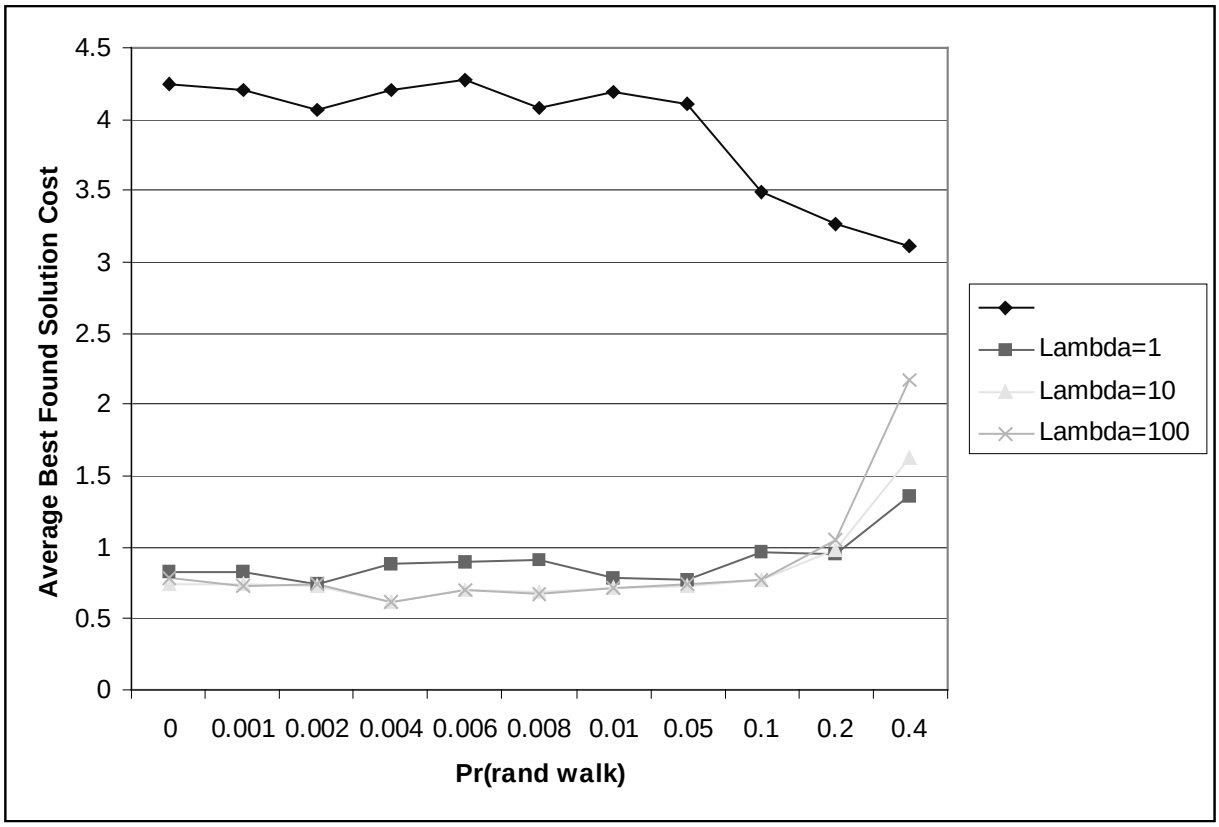


Figure 4-3: Results of the tuning phase for GLSSAT with random walk

Figure 4-3 shows the variation in performance of GLSSAT with different values of λ and $P_{randwalk}$ (the probability of picking an unsatisfied clause at random and then picking a variable in that clause at random and flipping it). This shows that when $P_{randwalk}$ is in the range [0.1,0.4], the performance is GLSSAT is improved, when λ is 0.1, with the average best found solution, having roughly 1 less unsatisfied clause. When λ is 1, 10, 100 and $P_{randwalk}$ >= 0.05, performance becomes increasingly worse with increasing $P_{randwalk}$.The best performing value of $P_{randwalk}$ appears to be 0.1, as this gives a performance increase when λ is 0.1, without compromising the performance too much for higher values of λ.

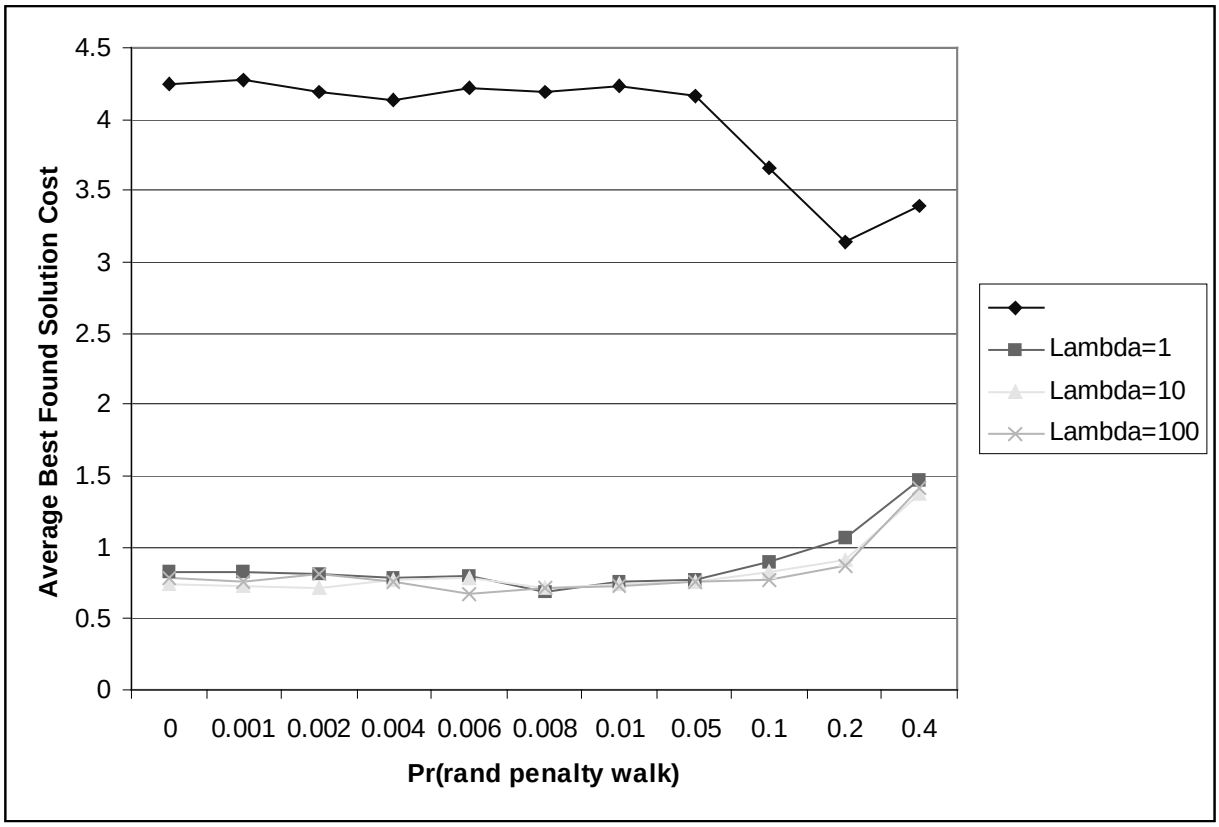
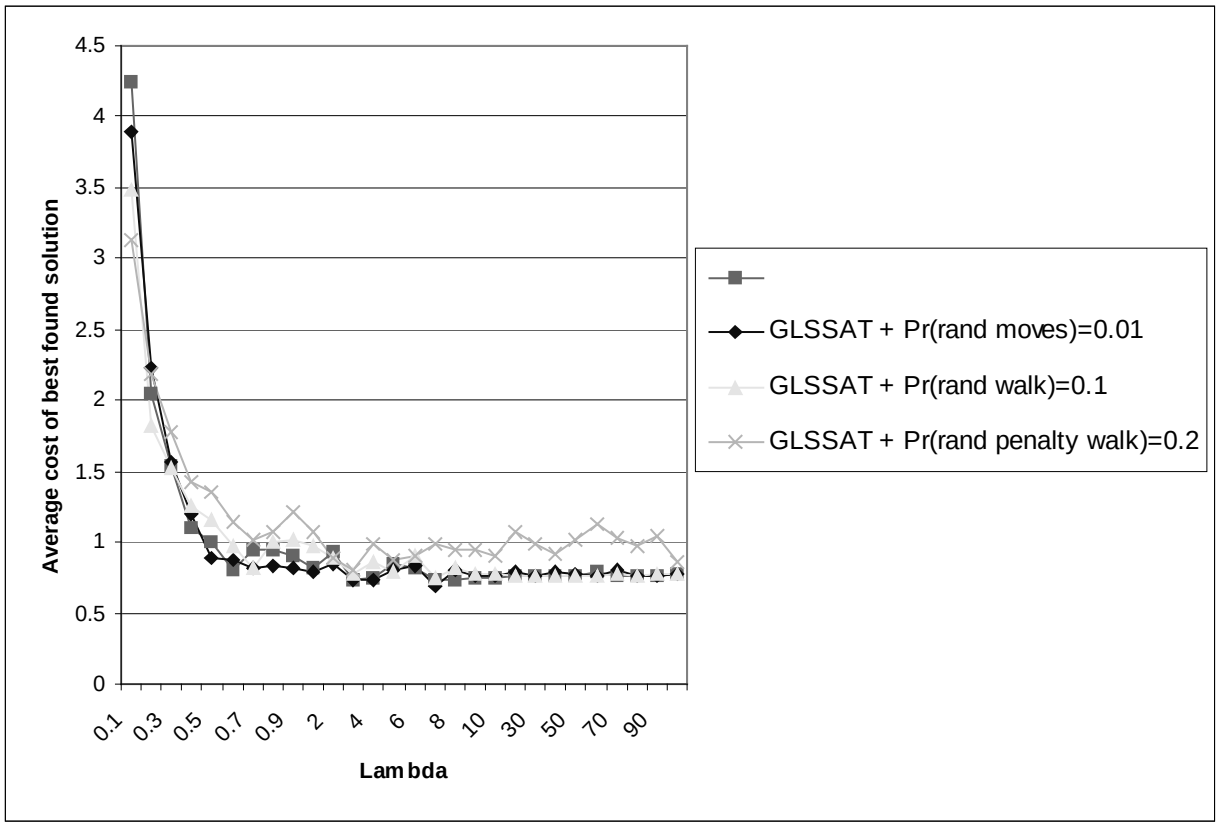
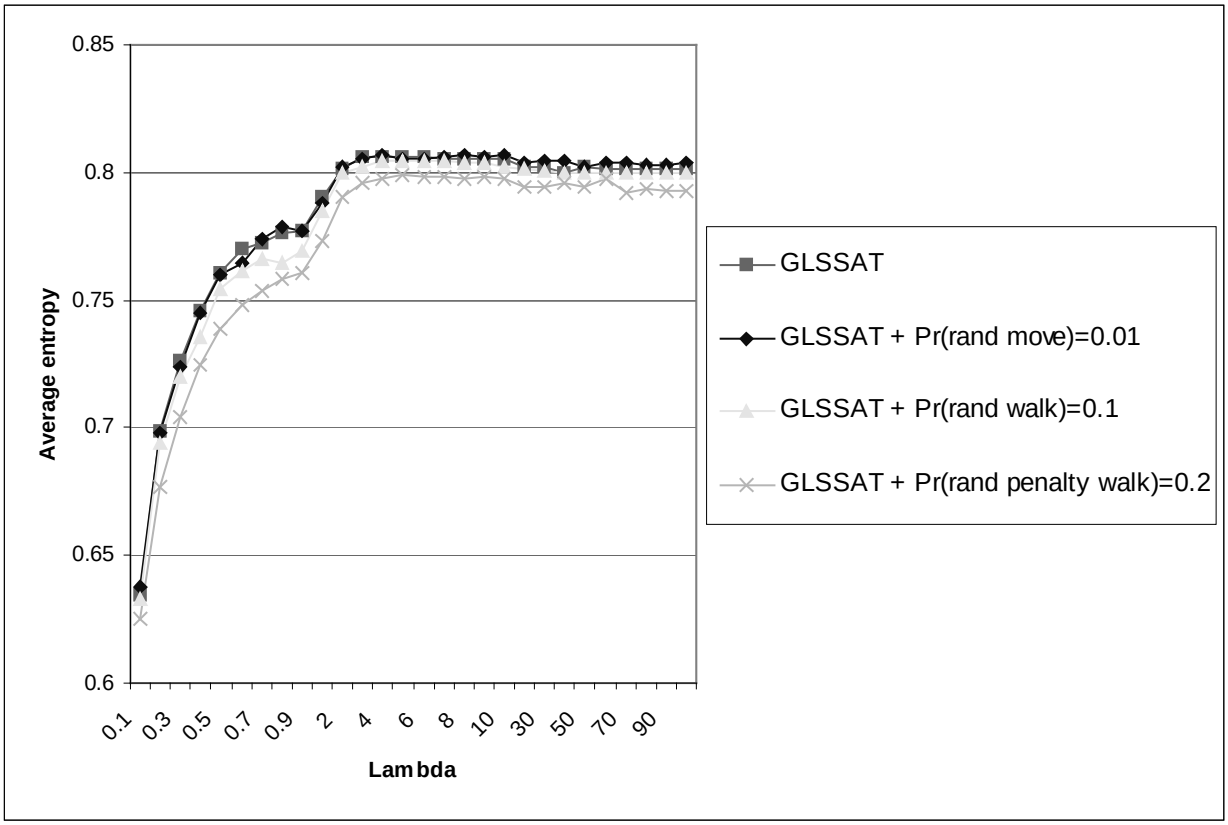
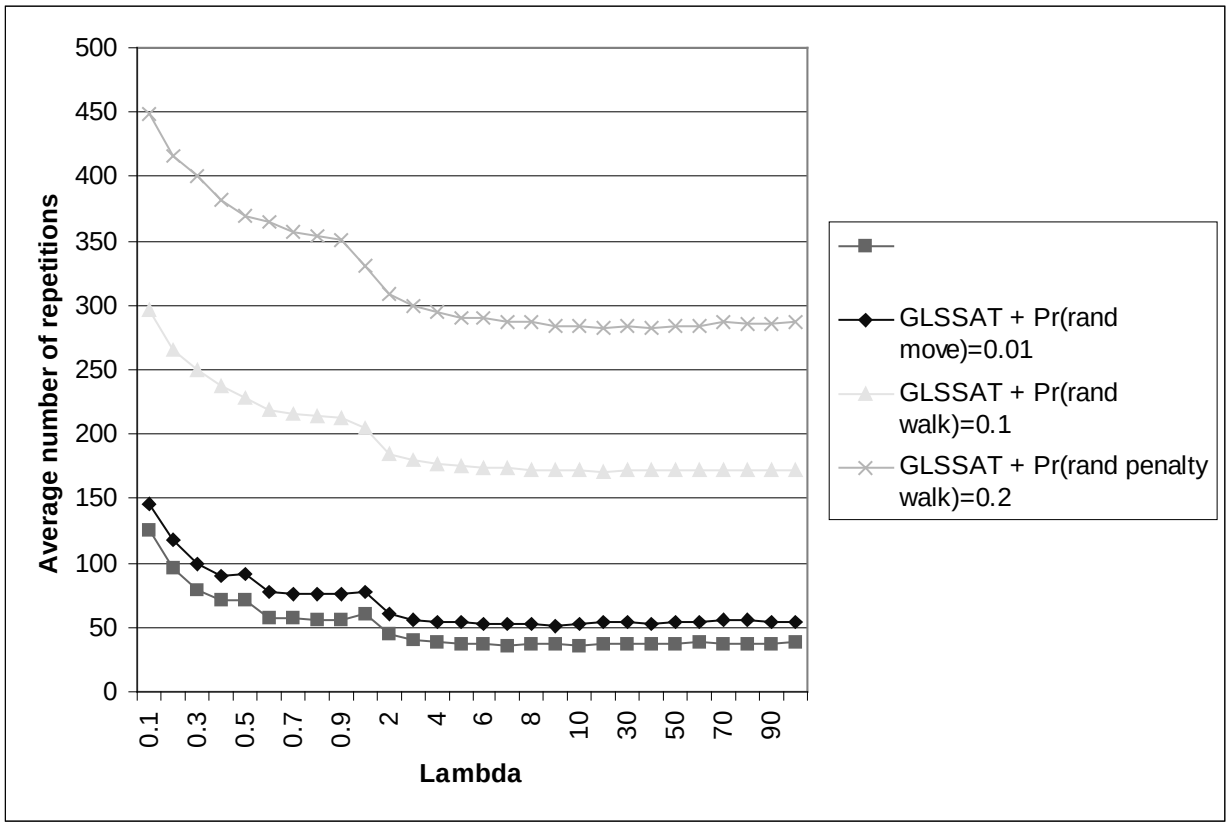


Figure 4-4: Results of the tuning phase for GLSSAT with random penalty walk

Figure 4-4 shows the performance of GLSSAT, with varying amounts of random penalty walk and varying values of λ. When the probability of making a random penalty walk move is greater than or equal to 0.4, then the performance of GLSSAT is degraded for larger values of λ. The best value for the probability of making a random penalty walk move appears to be $P_{randpenaltywalk} = 0.2$, as this gives maximum increase in performance when λ is 0.1, and does not degrade the performance too much when λ is higher.


4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
Average cost of best found solution
0.1
0.3
0.5
0.7
0.9
2
4
6
8
10
30
50
70
90
Lambda
GLSSAT + Pr(rand moves)=0.01
GLSSAT + Pr(rand walk)=0.1
GLSSAT + Pr(rand penalty walk)=0.2


0.85
0.8
0.75
0.7
0.65
0.6
Average entropy
0.1
0.3
0.5
0.7
0.9
2
4
6
8
10
30
50
70
90
Lambda
GLSSAT
GLSSAT + Pr(rand move)=0.01
GLSSAT + Pr(rand walk)=0.1
GLSSAT + Pr(rand penalty walk)=0.2


Average number of repetitions
500
450
400
350
300
250
200
150
100
50
0
0.1
0.3
0.5
0.7
0.9
2
4
6
8
10
30
50
70
90
Lambda
GLSSAT + Pr(rand move)=0.01
GLSSAT + Pr(rand walk)=0.1
GLSSAT + Pr(rand penalty walk)=0.2

4.4.2 MAX-SAT Results

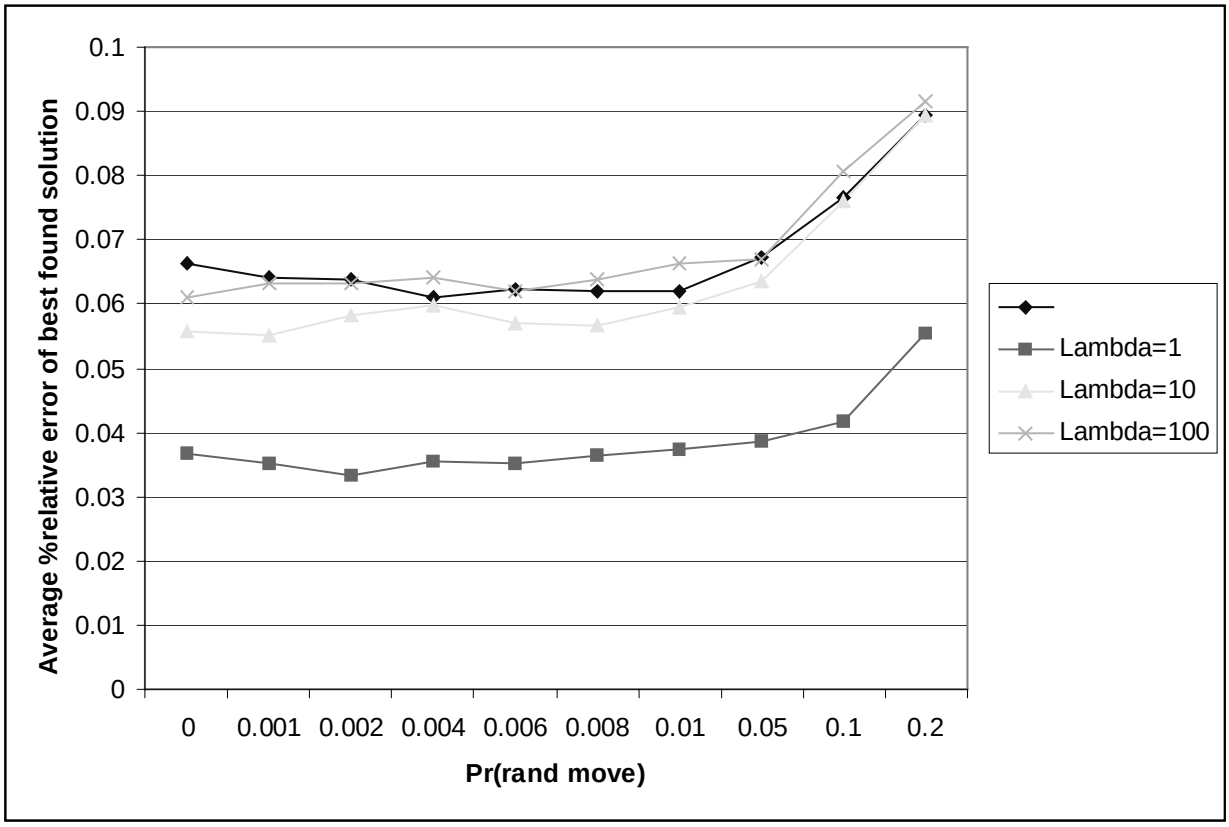


Figure 4-8: Results of tuning phase of GLSMAXSAT with random moves

Figure 4-8 shows the performance of GLSMAXSAT with random moves, with different probabilities of making a random move and different values of λ. It can be seen that values for the probability of making a random move greater than 0.05 produce worse performance, for all the values of λ used. We can see that 0.006 seems to give the best overall results, giving a very slightly better performance when λ is 0.1 and not degrading the performance too much for higher values of λ.

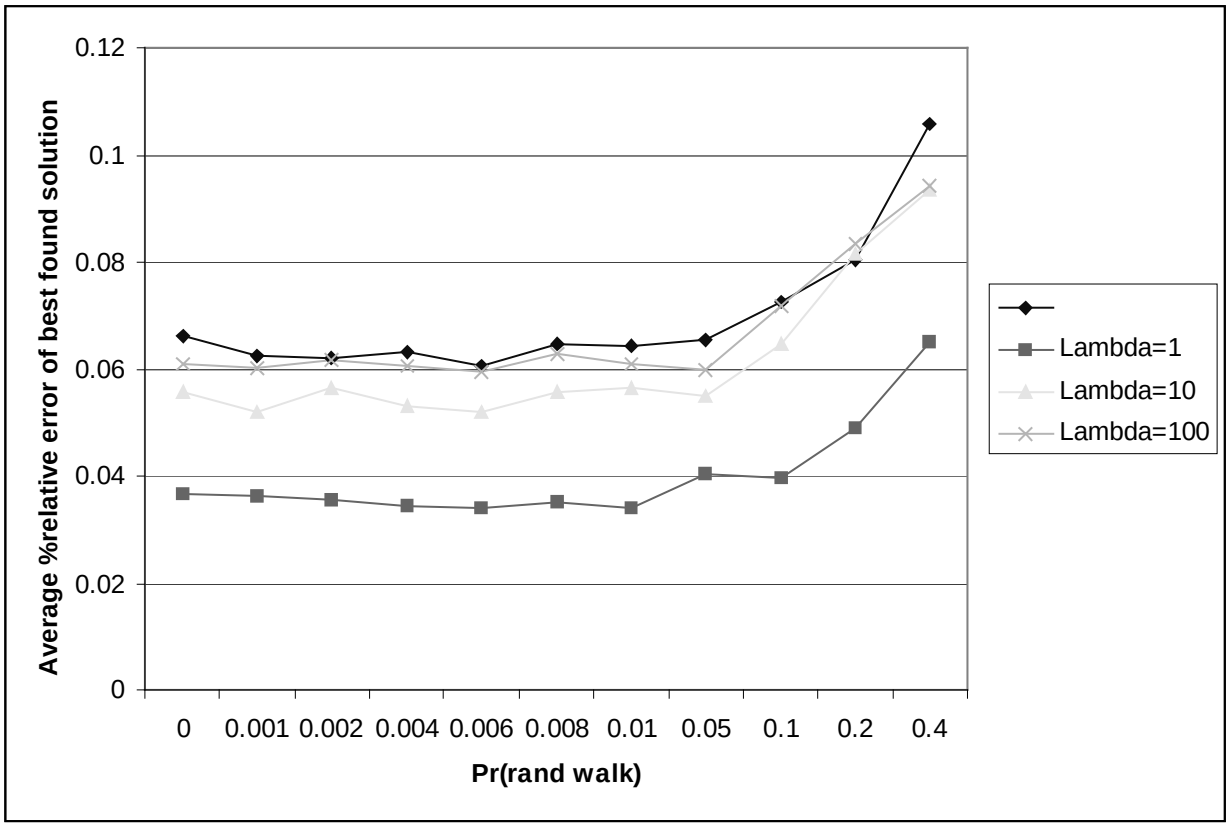


Figure 4-9: Results of tuning phase of GLSMAXSAT with random walk

Figure 4-9 shows the performance of GLSMAXSAT with random walk, for different values of λ and different probabilities for executing a random walk move. We can see that probabilities of picking a random walk move of 0.1, 0.2 and 0.4 are increasingly detrimental to the average performance of GLSMAXSAT. Smaller probabilities (less than 0.1), seem to work better, although none appear to give much performance increase. The best overall probability for picking a random walk move for GLSMAXSAT appears to be 0.006, as it gives a very slight increase in performance over the basic GLSMAXSAT for all values of λ tried.

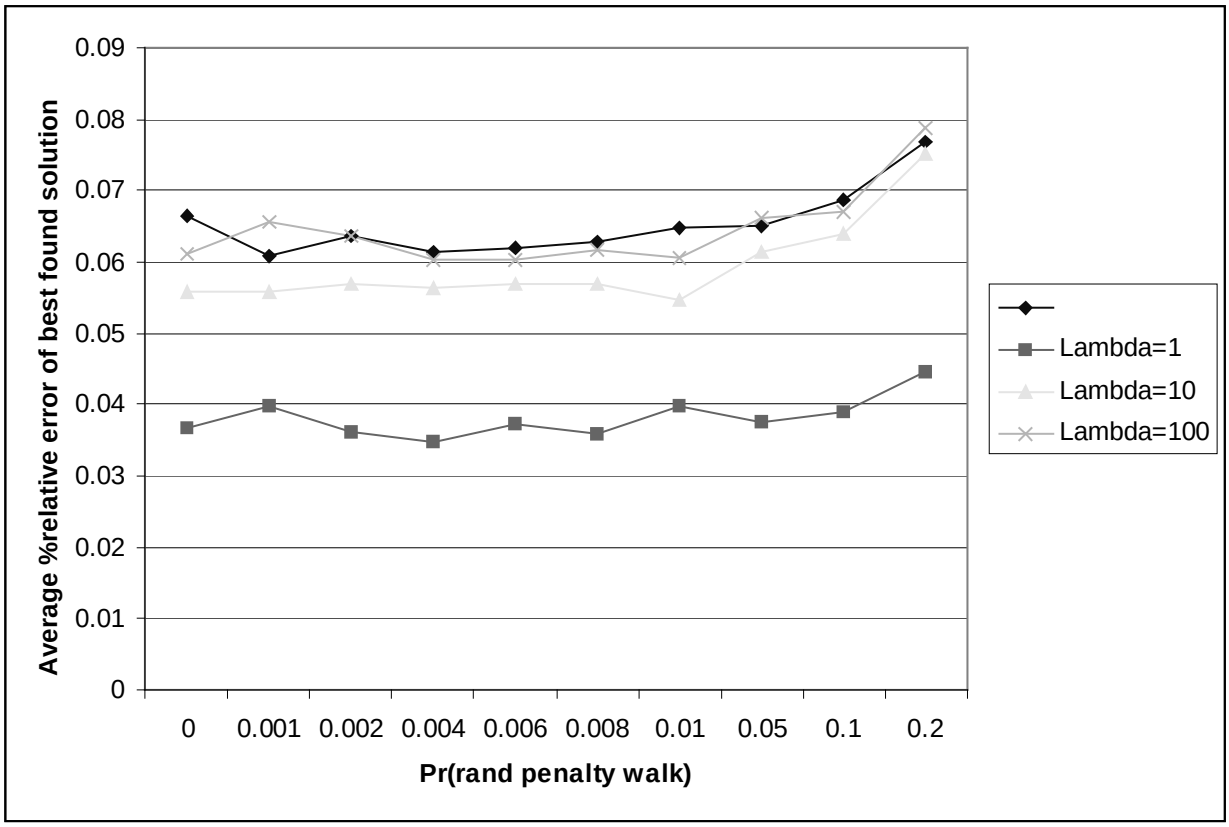


Figure 4-10: Results of tuning phase of GLSMAXSAT with random penalty walk

Figure 4-10 gives the performance of GLSMAXSAT for different values of λ and different probabilities of picking a random penalty walk move. This shows that probabilities of picking a random penalty walk move of 0.1 or more are detrimental to the performance of GLSMAXSAT. Lower probabilities than this produce slightly improved results when λ is 0.1, but not much (if any) improvement for the other λ settings tried. The best overall probability for picking a random penalty walk move appears to be 0.004, as this gives some improvement over the basic GLSMAXSAT when λ is 0.1, whilst not making the performance too much worse for the other values of λ.

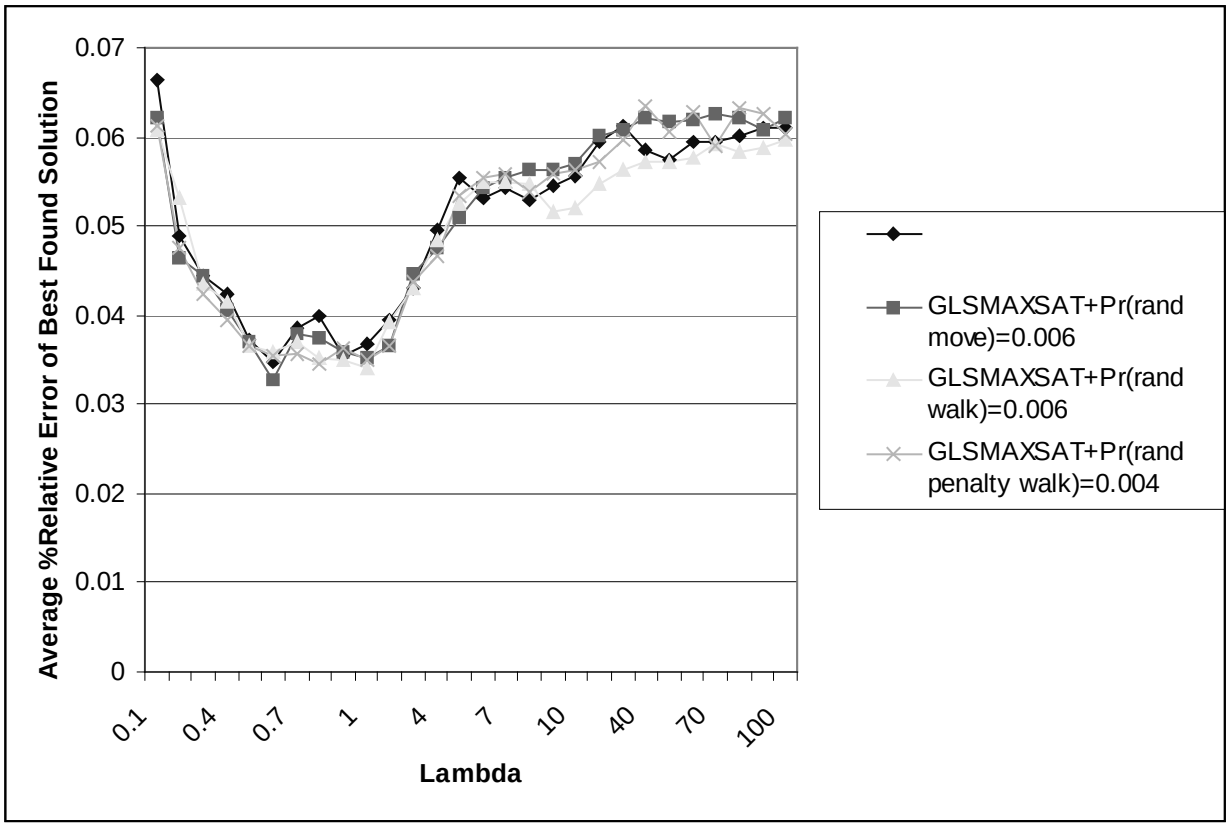


Figure 4-11: Average %relative error of best-found solutions for GLSMAXSAT versus GLSMAXSAT with random moves, random walk and random penalty walk

Figure 4-11 shows the average performance of GLSMAXSAT for each of the randomness schemes, over different values of λ, with the probability of using a particular scheme, fixed to the approximately tuned values found in the earlier experiments. From these results, we can see that there are only slight increases in performance for each of the schemes over the basic GLSMAXSAT, although these are offset by decreases in performance. Only random walk shows any consistent improvement over basic GLSMAXSAT when λ is between 10 and 100 (in fact, a sign test showed this was on average, a statistically significant improvement over all the

lambda settings, whilst the other two schemes gave on average statistically indifferent results from the basic GLSMAXSAT: see the appendix for details).

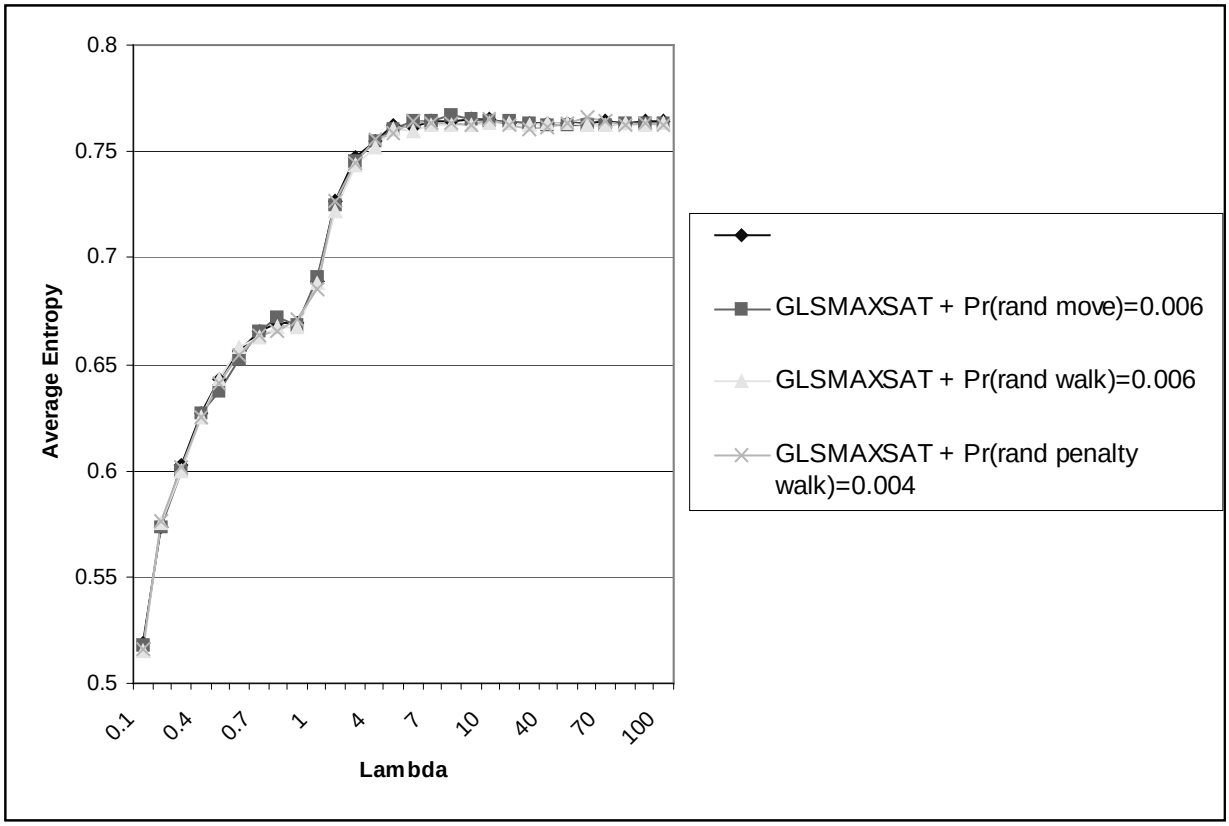


Figure 4-12: Average entropy for GLSMAXSAT versus GLSMAXSAT with random moves, random walk and random penalty walk

Figure 4-12 shows average entropy of the search for the different schemes added to GLSMAXSAT over a number of different values of λ. From this graph, we can see that the entropy values are approximately the same for all the schemes (although a sign test revealed that overall GLSMAXSAT with random moves produced statistically indifferent results for entropy, from the basic GLSMAXSAT, while the other two schemes produced on average statistically worse results for entropy). Perhaps the reason for this similarity in results for entropy, is due to the probabilities

for each scheme being so low, meaning that none have very much effect on the diversity of the search.

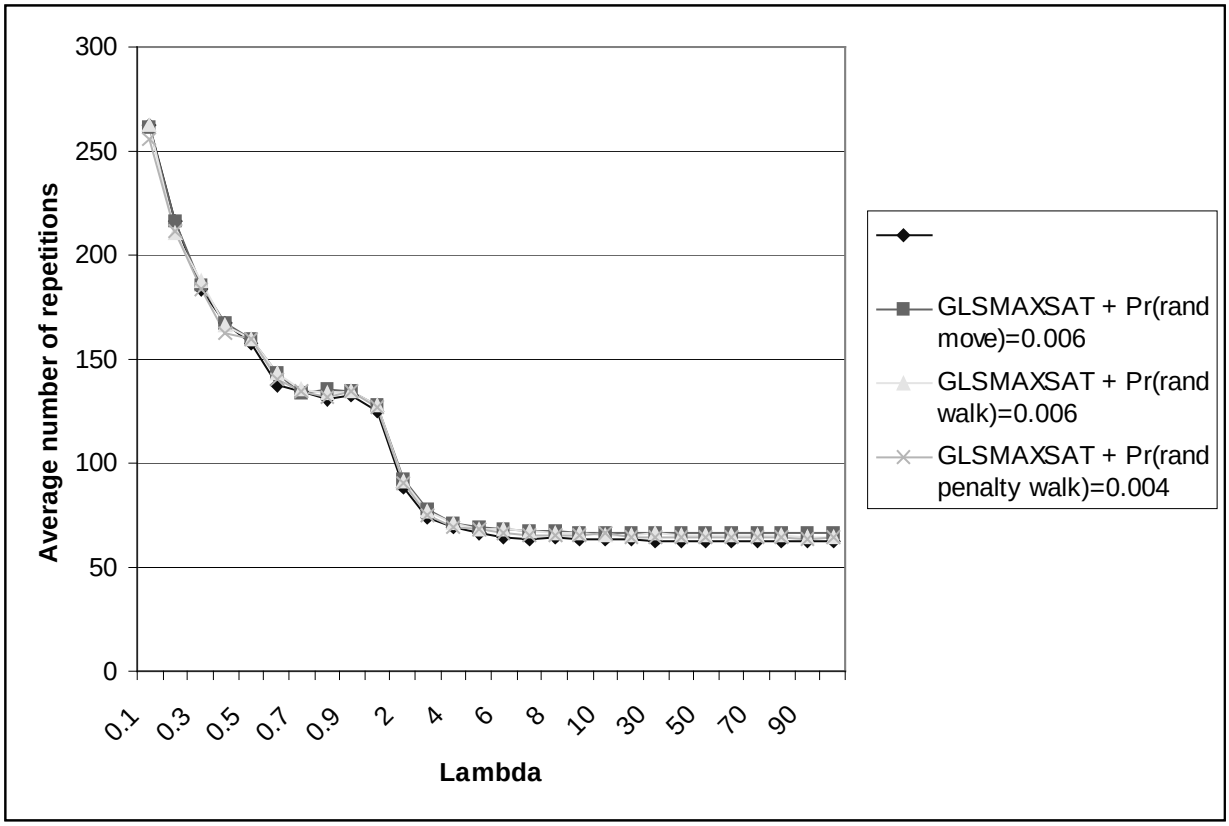


Figure 4-13: Average number of revisited solutions for GLSMAXSAT versus GLSMAXSAT with random moves, random walk and random penalty walk

Figure 4-13 shows the average number of solutions revisited during the search for the different random move schemes when added to GLSMAXSAT. Again, it appears there is little difference between them and between the basic GLSMAXSAT. However, from closer inspection of the data from which the graph is drawn, it appears that, overall, random moves give the most repeated solutions, then random walk, then random penalty walk and then the basic GLSMAXSAT. A sign test on the series in the graph revealed that overall all three schemes produced on average statistically significantly more revisited solutions than the basic GLSMAXSAT.

4.4.3 QAP Results

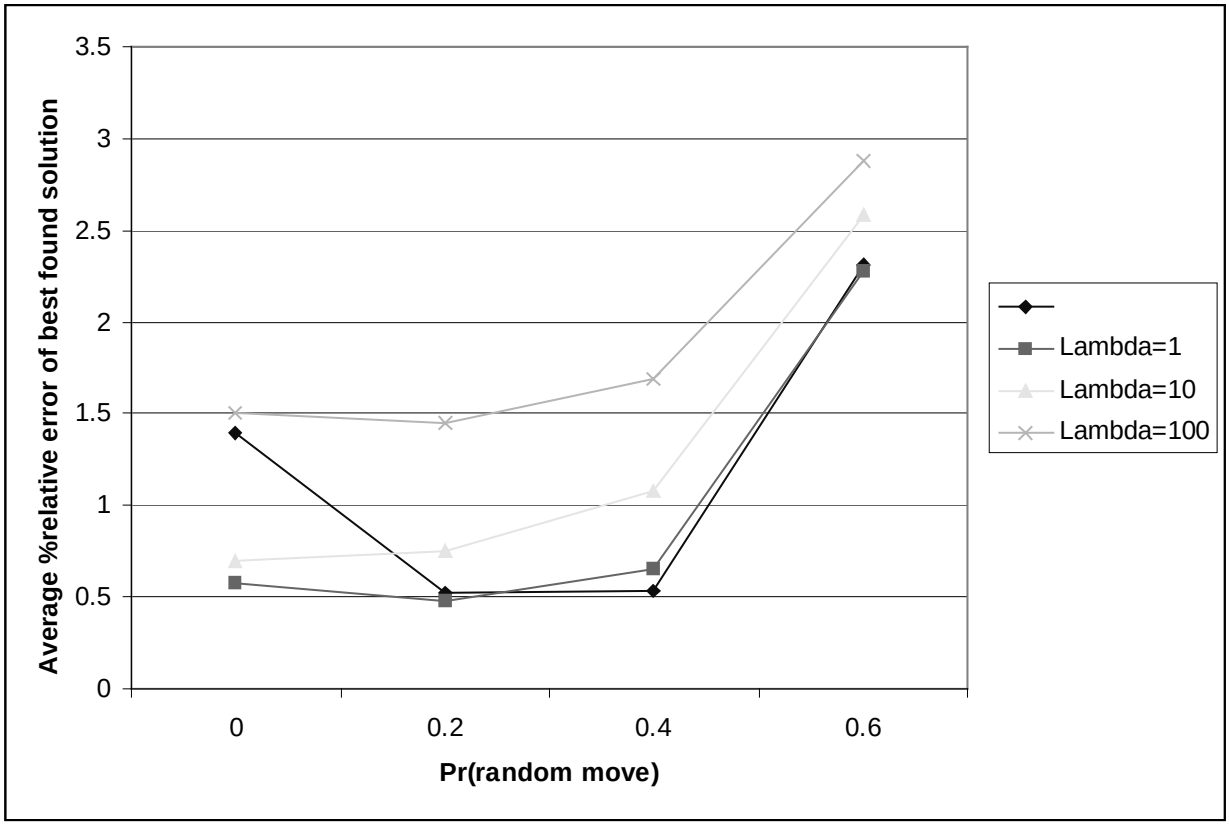


Figure 4-14: Results of tuning phase for GLSQAP with and without random moves

Figure 4-14 shows the performance of GLSQAP over different settings of λ and different probabilities for executing a move selected at random from the search neighbourhood. We can see that when a probability of 0.2 or 0.4 is used, that the performance of GLSQAP is improved for a λ setting of 0.1, but made slightly worse, when λ is 1, 10 and 100, when a probability of 0.4 is used. A probability of picking a random move of 0.6, gives worse performance over all settings of λ tried. The best overall setting for the probability of picking a random move, for GLSQAP, appears to be 0.2, as this gives maximum increase in performance when λ is 0.1, 1 & 100 and minimum decrease in performance when λ is 10.

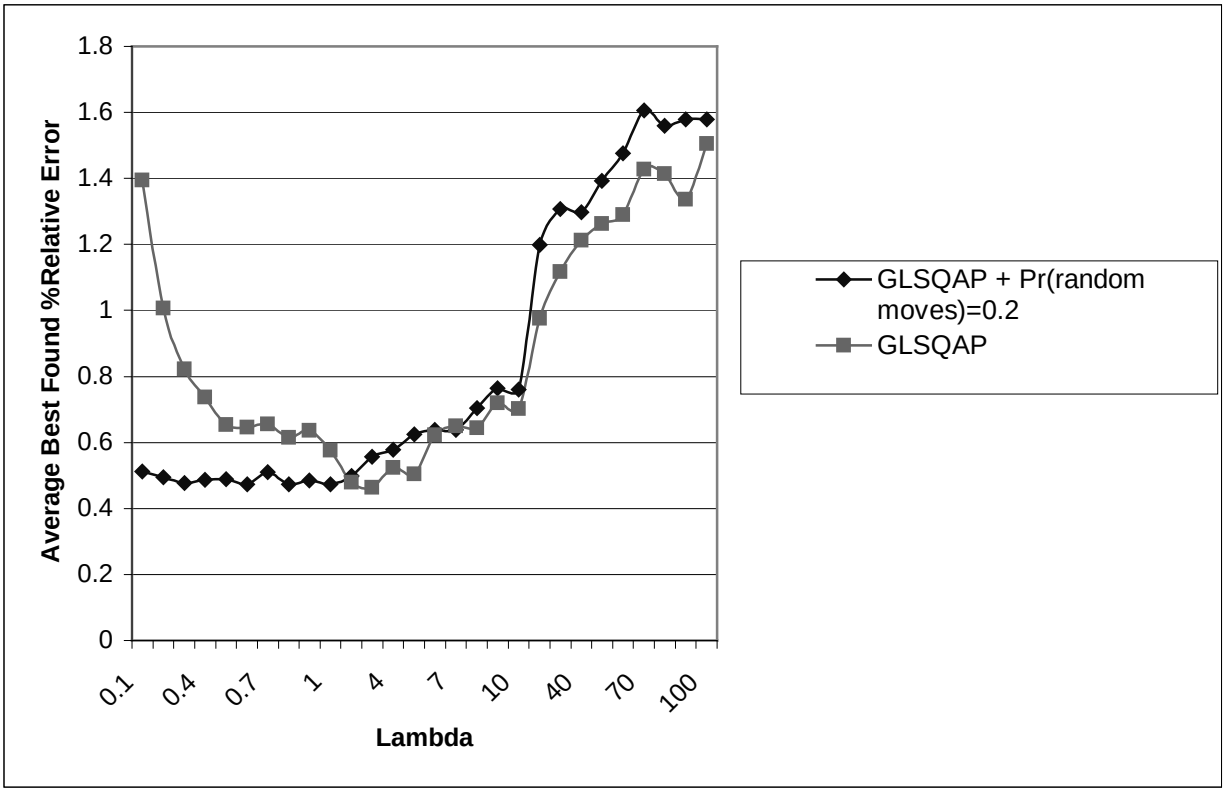
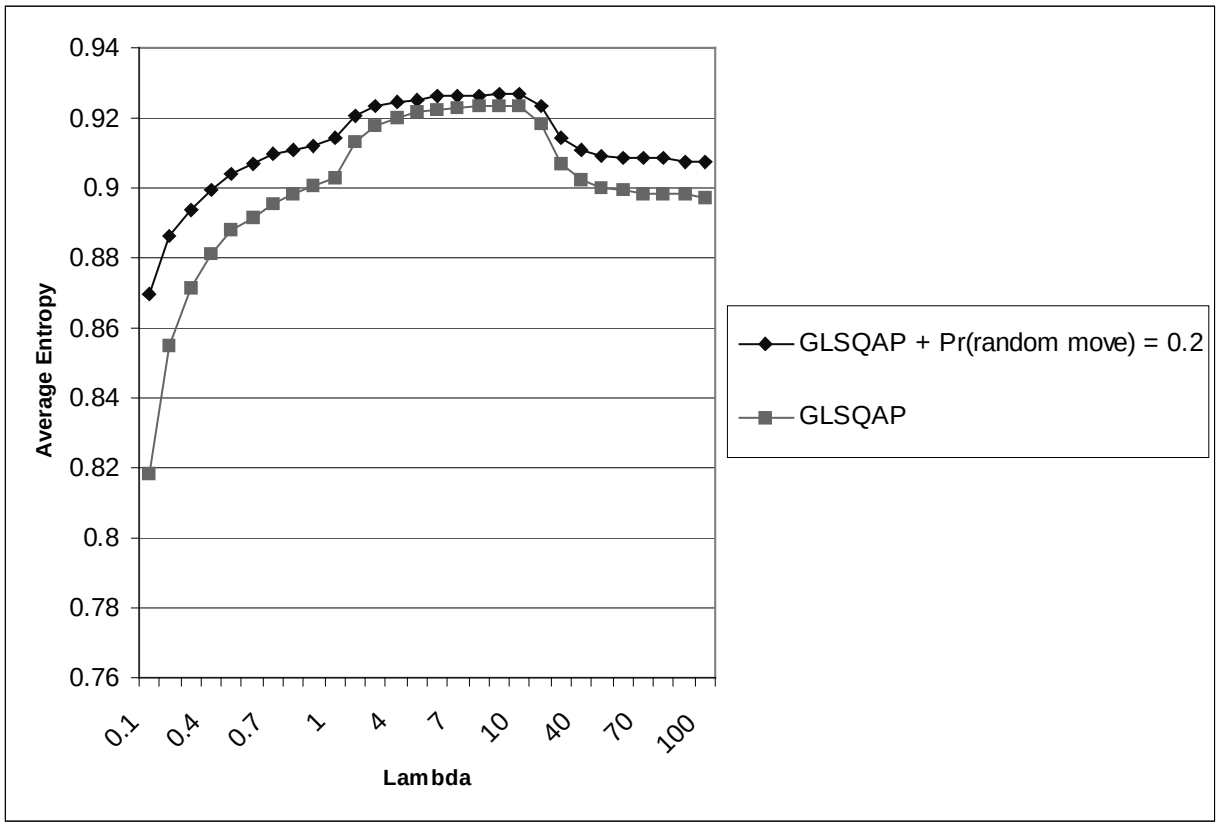
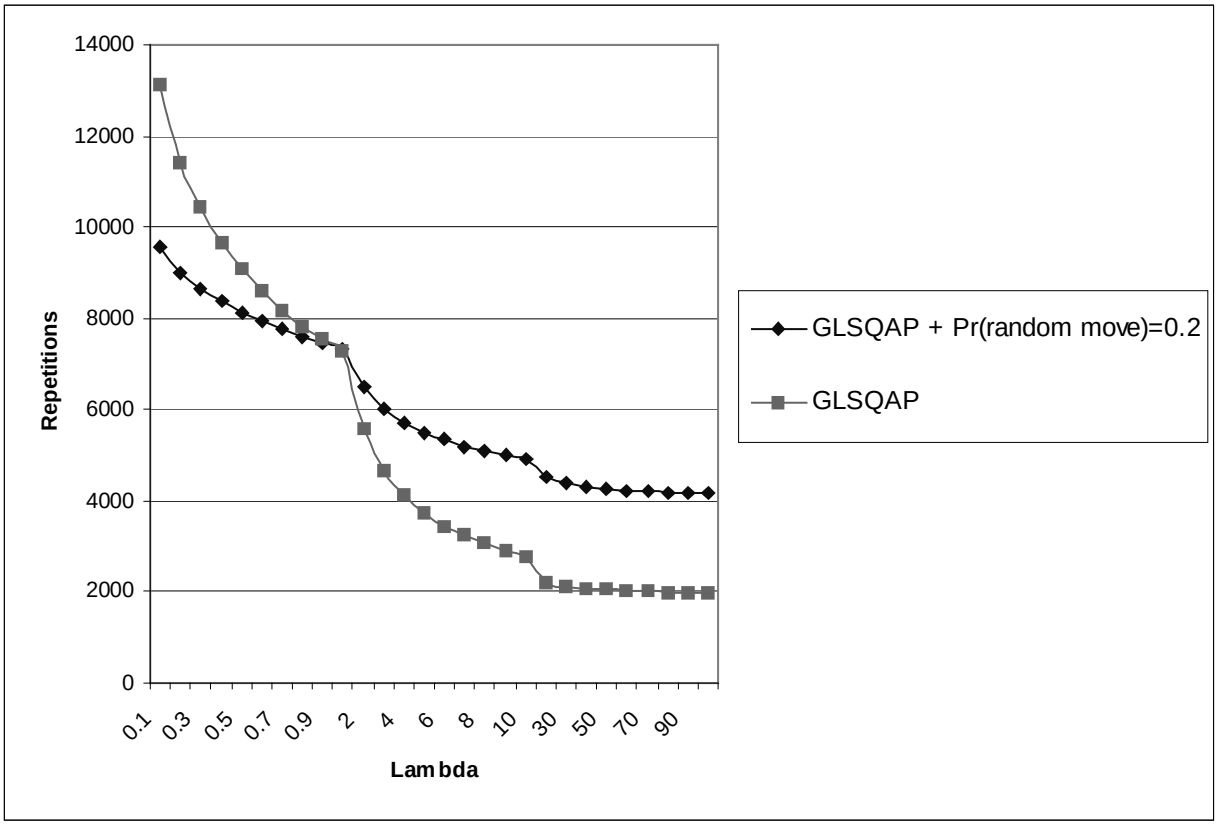


Figure 4-15: Average %relative error of best found solutions of GLSQAP versus GLSQAP with random moves

Figure 4-15 shows the performance of GLSQAP with and without random moves over a range of different λ settings, with the probability of making a random move fixed at 0.2. We can see that the performance of GLSQAP is much better with random moves for λ settings between 0.1 and 1, with the performance with random moves then becoming gradually worse than without random moves, as λ increases from 1 to 100.


0.94
0.92
0.9
0.88
0.86
0.84
0.82
0.8
0.78
0.76
Average Entropy
0.1
0.4
0.7
1
4
7
10
40
70
100
Lambda
GLSQAP + Pr(random move) = 0.2
GLSQAP


14000
12000
10000
8000
6000
4000
2000
0
Repetitions
0.1
0.3
0.5
0.7
0.9
2
4
6
8
10
30
50
70
90
Lambda
GLSQAP + Pr(random move)=0.2
GLSQAP

4.5 Comparison

Overall, the performance of all the random move variants tried gave little increase in performance for the SAT problem, except for very small values of λ (Figure 4-5). For the weighted MAX-SAT problem, there was a slight increase in performance when random walk was used, for λ settings between 10 and 100, although otherwise the performance of the random move variants was about the same as without the extensions (Figure 4-11). For the QAP, the performance was greatly improved, using random moves, for λ settings between 0.1 and 1, removing the decrease in performance (present without random moves in GLSQAP) from using values of λ which are too low, although the performance was slightly worse for λ settings from 1 to 100 (Figure 4-15).

4.5.1 The SAT Problem

The reason for the worse or indifferent performance of random moves, random walk and random penalty walk for GLSSAT may be because the majority of the search effort when solving the SAT problem is spent exploring plateaus of solutions of equal cost (see Figure 4-18 & Figure 4-19, showing this by plotting solution cost over a run of GLSSAT, as previously done by Selman & Kautz in [77] with GSAT. In addition to this, the reader interested in problem landscapes should refer to [27,61,70,71,72,73,74]). However, the size of λ is less critical when exploring plateaus. This is because any small increase in the augmented objective function is enough to escape from the current solution, and encourage a move to a neighbouring unpenalised solution with equal cost (but due to penalties, lower augmented cost). Adding random moves, random walk or random penalty walk, only seems to decrease

the efficiency of the search, by increasing the number of revisited solutions, and in some cases decreasing the diversity of the search.

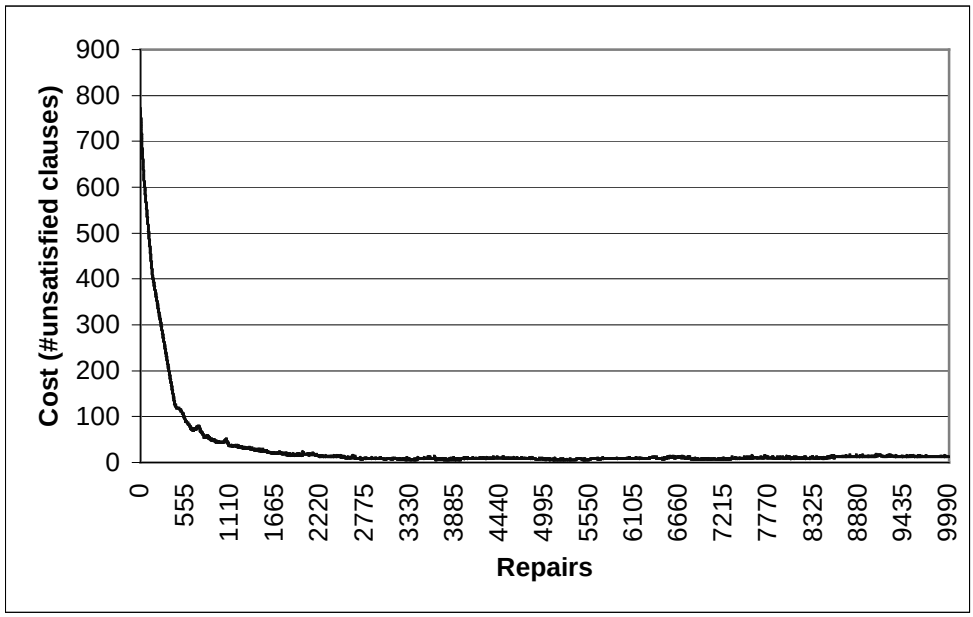


Figure 4-18: A graph of solution cost over the first *10,000 repairs* over 1 run of basic GLSSAT for problem instance ssa7552-038

This is backed up by the results in Figure 4-6 and Figure 4-7, which show that the entropy, strongly related to the diversity of the search, is decreased by random move schemes for the SAT problems, and the number of revisited solutions is increased. One possible reason why this might be is that a random move (or random walk move, or random penalty walk move) may be made and then immediately undone, so that most times such a move is made, the previous solution is revisited after that move has been reversed. This would explain the increase in the number of repeated solutions, and also explain the decrease in the average label entropy in some cases, as more solutions will be visited with the same labels.

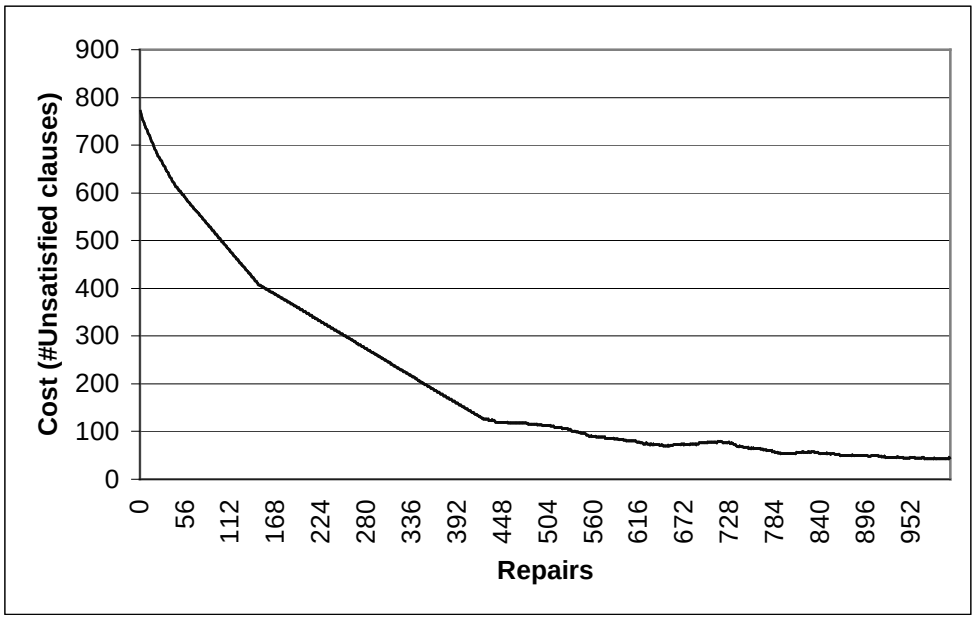


Figure 4-19: Graph of solution cost over first *1000 repairs* over 1 run of GLSSAT for problem instance ssa7552-038

4.5.2 The MAX-SAT Problem

The results for the MAX-SAT problem are similar to SAT, although random walk gives a slight improvement in performance for high values of λ, possibly by helping it go through barriers to the local search caused by penalties. Apart from this, we believe that a similar hypothesis as that advocated in section 4.5.1 would explain why the random move extensions do not generally improve the performance of GLSMAXSAT, especially since GLSMAXSAT actually uses the number of unsatisfied clauses as the objective function, rather than weighted sum of unsatisfied clauses (which would be the natural choice of objective function), and only uses the weights of clauses when deciding which features (clauses) to penalise. It should be noted that the differences, if any, in the graphs for entropy and repeated solutions are much smaller for the MAX-SAT than for the SAT problem. This is probably due to the smaller probabilities (0.006 compared to 0.01, for $P_{randmove}$, 0.006 compared to 0.1

for $P_{randwalk}$ and 0.004 compared to 0.2 for $P_{randpenaltywalk}$) of making a random move (or random walk move, or random penalty walk move) used for the experiments for the MAX-SAT problem.

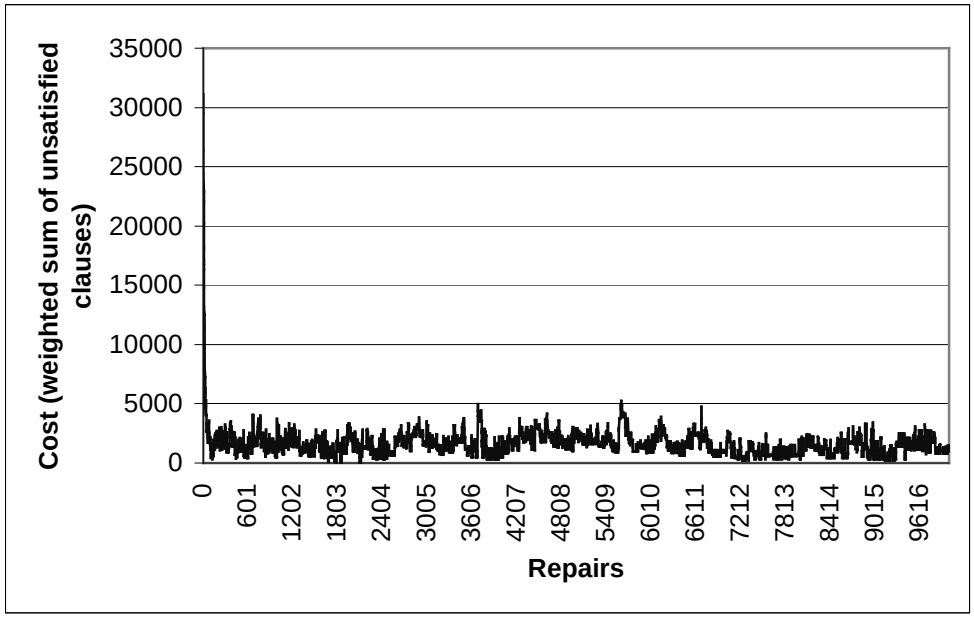


Figure 4-20: Graph of solution cost over first *10,000 repairs* over 1 run of GLSMAXSAT for problem instance jnh208

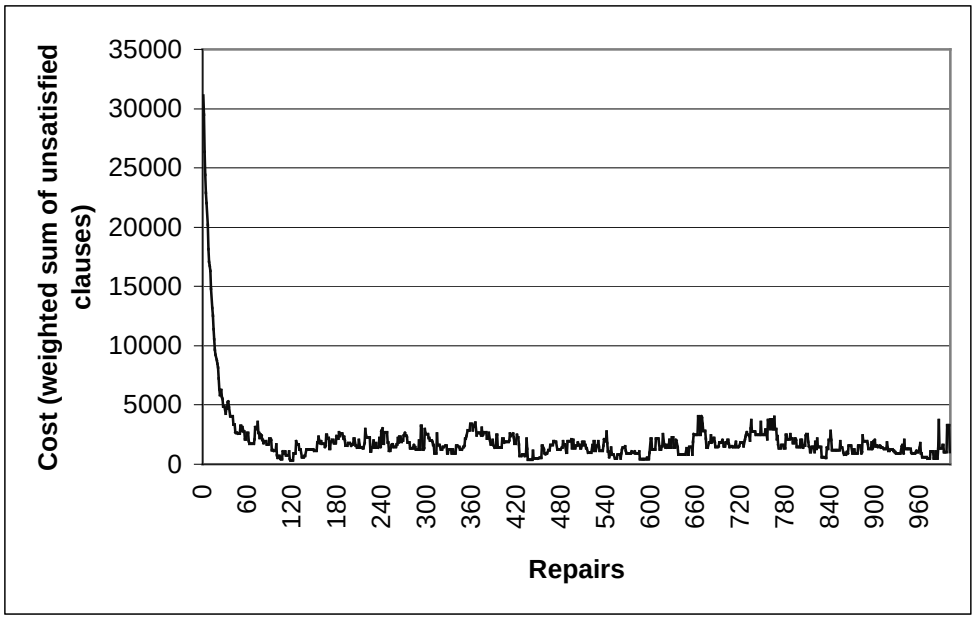


Figure 4-21: Graph of solution cost over first *1000 repairs* over 1 run of GLSMAXSAT for problem instance jnh208

4.5.3 The QAP Problem

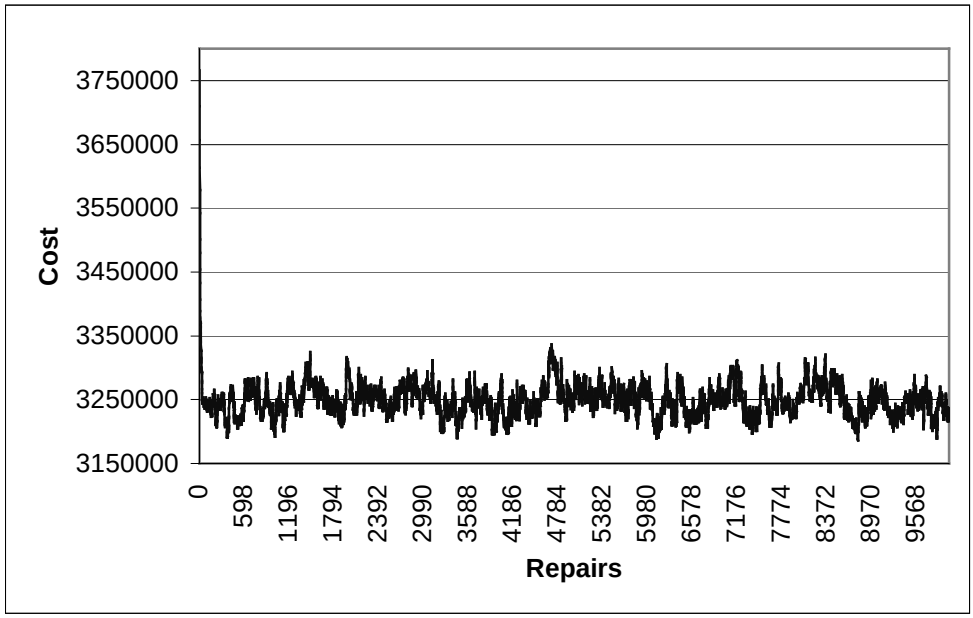


Figure 4-22: Graph of solution cost over first *10,000 repairs* over 1 run of GLSQAP for problem instance tai40a

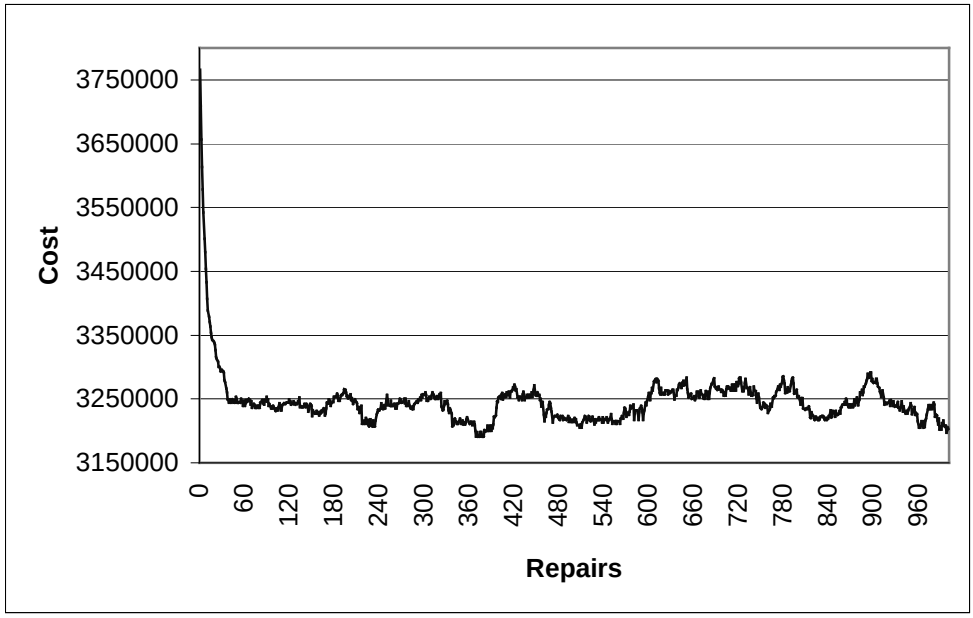


Figure 4-23: Graph of solution cost over first *1000 repairs* over 1 run of GLSQAP for problem tai40a

We believe that the improvement in performance for GLSQAP with random moves, when λ is small (between 0.1 and 1), is because such moves help diversify the search,

allowing it to escape more quickly from basins in the search landscape. Unlike the SAT and MAX-SAT problems, the QAP landscape does not mainly consist of long plateaus, but is composed of many basins at various different levels of cost (see Figure 4-22 & Figure 4-23 showing an example of this, by plotting the cost of solutions visited over a run of GLSQAP). This is an important difference between the SAT and MAX-SAT problems and the QAP problem, because it means that low values of λ become ineffective for solving these problems, as it takes more iterations of GLSQAP to escape from these basins if λ is set too low. Thus when random moves are added to GLSQAP, they provide an alternative means for the local search algorithm to escape from these basins, and thus helping diversify the search when λ is too low. This is backed up by Figure 4-16, which shows that the entropy of the search (which is a measure of the diversity of the search) is higher for all values of λ for GLSQAP with random moves, than without and, in particular, is higher for small values of λ. Even more interestingly, Figure 4-17 shows that the average number of revisited solutions, when λ is between 0.1 and 1, is lower for GLSQAP with random moves than without, but then crosses over when λ is 1. The number of repetitions for GLSQAP with random moves then becomes higher, when λ is between 1 and 100. At the same point the performance of GLSQAP (see Figure 4-15) also switches from being better with random moves, to be being worse, when λ is between 1 and 100. This shows that when λ is too low, the addition of random moves to GLSQAP helps diversify the search, but when λ becomes sufficiently large, then the addition of random moves may actually cause a slightly less efficient search (resulting in more repeated solutions). This evidence backs up the theory that random moves help GLS diversify the search in the QAP, when too low a setting of λ hinders the ability of GLS to do so without them.

4.6 Discussion

4.6.1 When can random moves help GLS?

We have shown in the previous sections, that random moves are quite helpful in solving the Quadratic Assignment Problem when a low setting of λ is used, but not so useful in solving problems such as the SAT and weighted MAX-SAT problems containing some plateau regions in the cost landscape. We believe that this result will generalise and that random moves will help GLS when it is faced with problems in general where many local minima basins and few plateaus are present in the cost landscape.

4.6.2 Future work

Due to time considerations there were a number of issues we would have liked to have investigated, but did not have time to do. These include running GLS on top of the walksat [79] local search algorithm, with some of the other extended walksat heuristics, such as Rnovelty and Novelty [59], to see if these would perform better than the random moves, random walk and random penalty walk schemes we tried. In addition, another variant we would have liked to have investigated, would have been to restrict making random moves to those variables in the SAT and MAX-SAT problems involved in unsatisfied clauses, in the same way that GSAT when random walk is used with it [79]. Another possible area for investigation would be to vary when and how random moves are made, in a similar way to Iterated Local Search [88] or Variable Neighbourhood Search [65]. This could be done by restricting when random moves can be made (for example only allowing random moves to be made when a local minimum has been reached) and also varying the "strength" of random moves (how much it changes the solution) by allowing a sequence of random moves

to be made, rather than just a single random move. It would have also been interesting to try some other smaller probabilities for executing random moves for GLSQAP, to see if these might work even better than 0.2, as we did in the SAT and MAX-SAT problems, and also experiment with even smaller λ settings for the SAT and MAX-SAT to see if GLS with random moves helps more for these settings.

As well as trying different parameter and heuristic combinations, it would also be worth applying GLS with random moves to other problems, particularly those with landscapes similar to the QAP (possibly the Travelling Salesperson Problem), with many local minima basins, but fewer plateaus. This would help verify the results presented in this chapter.

4.7 Conclusion

We have shown that random moves, when added to GLS, can be useful for the QAP problem, but not the SAT and MAX-SAT problems. We have presented a theory of why this might be. Our theory is that random moves help GLS escape from basins in the search landscape of the QAP. Such basins are not so common in the SAT and MAX-SAT problems, which consist mainly of plateaus, where GLS already performs very well, even with very low values for λ and without random moves. This possibly explains why random moves do not improve the performance of GLS in solving these types of problems. We therefore believe that random moves are a useful addition to the GLS user's toolkit for problems with very rugged cost landscapes, which contain many local minima or basins in the search landscape and where it is difficult to find a good setting for λ.

5 Combining Aspiration and Random moves

We have shown in previous chapters the effectiveness of adding aspiration and randomness separately to GLS. In this chapter, we study their combined effect. We also provide the results of some other algorithms varying their main parameter settings, to give the reader other algorithms with which to compare the parameter sensitivity of the extended GLS.

5.1 Motivation

We have observed that aspiration moves help GLS when λ is set too high and random moves help GLS when λ is set too low. By combining the extensions of GLS, the hope is that sensitivity of the algorithm's performance to the λ parameter may be reduced and that performance will be improved for some settings of λ, where that extension helps, without compromising the performance of the extended GLS for other settings of λ.

5.2 Experiments

To test whether combining aspiration moves and random moves with GLS would be successful, we ran GLS with both extension for each set of problems. We varied the λ parameter over the values:

- 0.1 to 1 in steps of 0.1,
- 1 to 10 in steps of 1 and
- 10 to 100 in steps of 10.

The reason for choosing these values was to examine the effect of small, medium and large changes to the λ parameter, as well as the effectiveness of small, medium and

large values of λ. In addition to this, we also used the experimental results available from previous chapters. We also ran another state-of-the-art algorithm for each set of problems, over a range of parameter settings, to give the reader another algorithm to compare the sensitivity of GLS with respect to λ.

To test how the extended GLSSAT compared with state-of-the-art algorithms, we ran GLSSAT over easier soluble DIMACS problems, allowing 10*n repairs (n = number of variables in the problem) and 10 runs per problem, over the range of λ values previously indicated. These experiments were performed with GLSSAT plus aspiration moves and random moves, GLSSAT plus aspiration moves and random walk, and GLSSAT plus aspiration moves and random penalty walk. We also ran Walksat (see [81,79] & chapter 2), under the same conditions, varying the noise parameter over the set of values:

- 0.01 to 0.1 in 0.01 steps and
- 0.1 to 1 in 0.05 steps.

These values were chosen to test the effect of slightly smaller changes to smaller values, as well as slightly larger changes to the higher values of Walksat's noise parameter (there seemed little point in making small changes which were insignificant compared to the parameter values being tested).

To test how the extended GLSMAXSAT compared to state-of-the-art algorithms, we similarly ran the GLSMAXSAT variants over all the max-sat benchmark problems, allowing 10*n repairs and 10 runs, over the set of λ values previously indicated. We ran this experiment for GLSMAXSAT plus aspiration moves plus random moves, random walk or random penalty walk. We also ran maxwalksat [43] (also see Chapter

2) over the same set of noise values as for walksat, also under the same conditions as for GLSMAXSAT.

Finally, to test the extended GLSQAP against state-of-the-art algorithms, we ran GLSQAP over all the small to medium sized ($10 \leq n \leq 40$) problems from the QAPLib benchmarks, allowing 1000**n* repairs and 10 runs per problem, with aspiration moves and random moves, with λ varied over the values previously indicated. To provide another algorithm for comparison, we ran robust tabu search [89,90] (also see Chapter 2) under the same conditions, varying its u parameter (the tabu list size = $u * n$ varied randomly over a run by +/- 10%) over the set of values (keeping the t parameter fixed at 3.5):

- 0.1 to 2 in steps of 0.1 and
- 2 to 10 in steps of 1.

We also varied the t parameter (the number of repairs a facility-location pair may be not present in a solution, before it is made a tabu condition to be made present) over the set of values (whilst keeping the u parameter fixed at 1):

- 0.1 to 1 in steps of 0.1,
- 1 to 10 in steps of 1 and
- 10 to 100 in steps of 10

As before, we chose these values to test the effects of making small changes to smaller values of the parameter, whilst making larger changes to larger values of the parameter, since there would have been little point in making changes which relative to the parameter value were extremely small.

5.3 Results

In this section we give the results of the experiments described above. For each graph, each line represents a particular algorithm and where there is more than one algorithm per graph, a legend shows which algorithm is which. For each statistic used, we plot the average over all runs for each parameter setting (it should be noted that the scale is not always linear).

5.3.1 SAT Results

5.3.1.1 Comparing GLSSAT variants

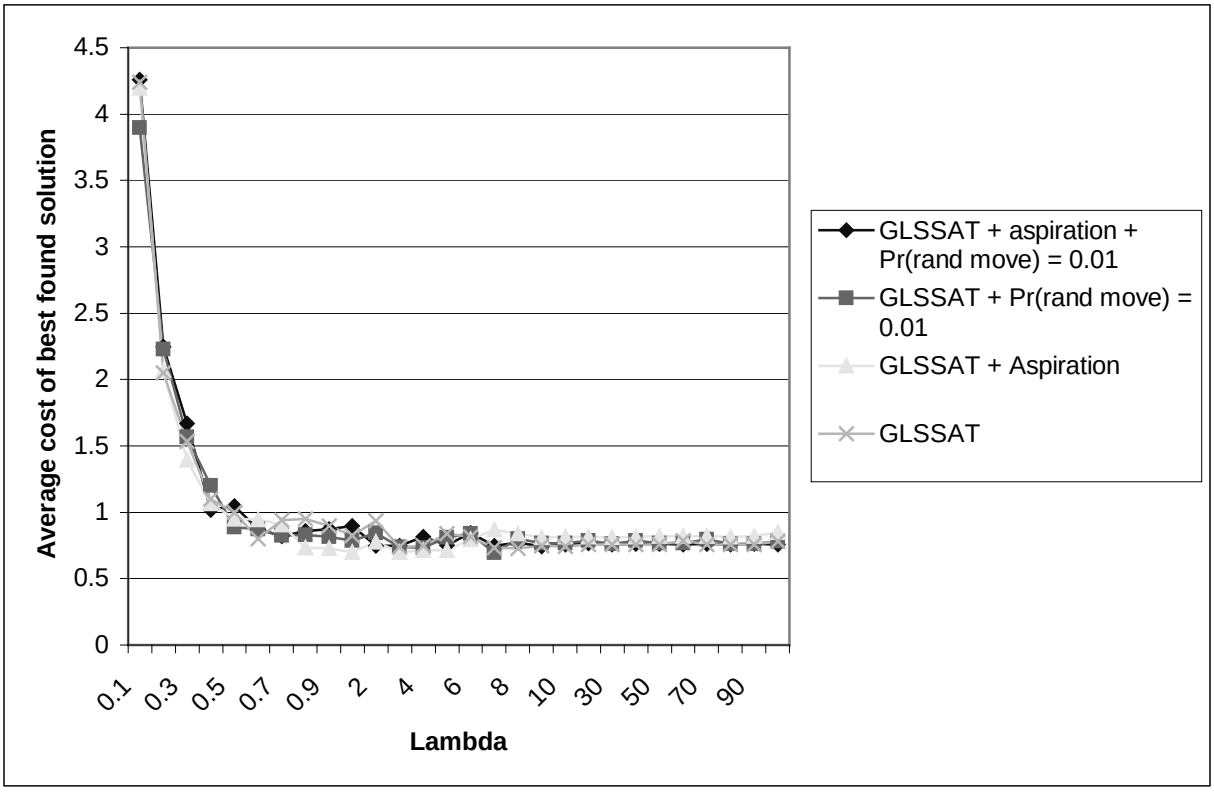


Figure 5-1: Performance of GLSSAT with random moves, with and without aspiration

Figure 5-1 shows the performance of the variants of the GLSSAT with and without random moves, over a range of λ settings. From this, it appears that there is little

difference between each of the variants, although GLSSAT with aspiration moves appears to be slightly better than the other variants for λ values between 0.8 and 5. The results appear inconclusive, however, with all the variants performing roughly the same overall, some performing better for some settings and some performing better for others.

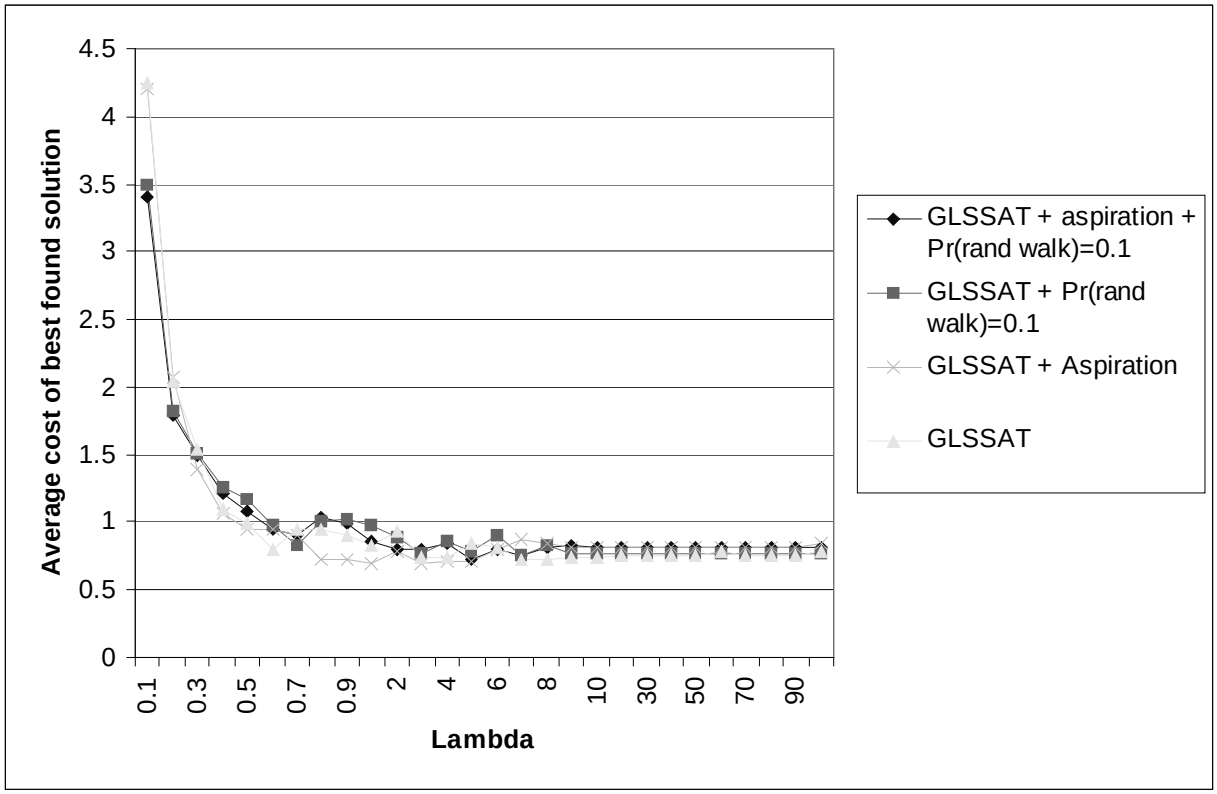
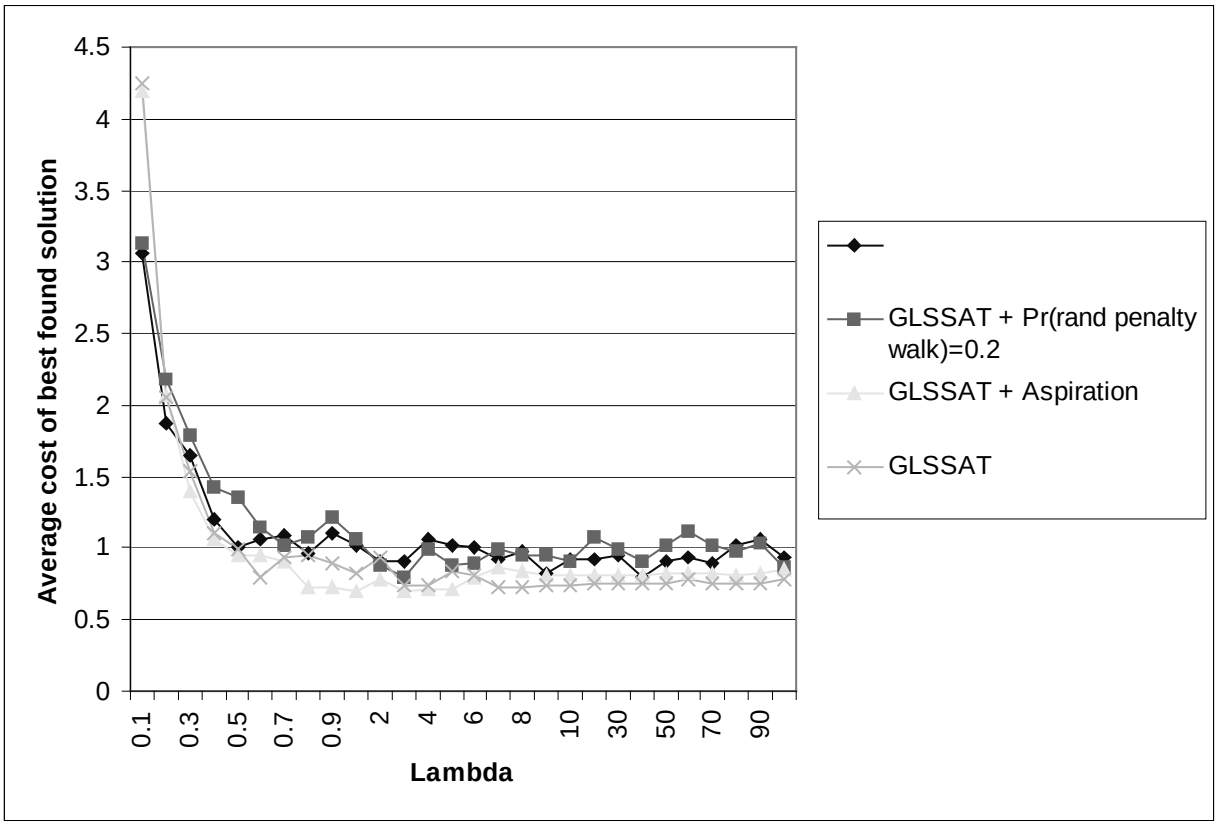


Figure 5-2: Performance of GLSSAT with random walk, with and without aspiration moves

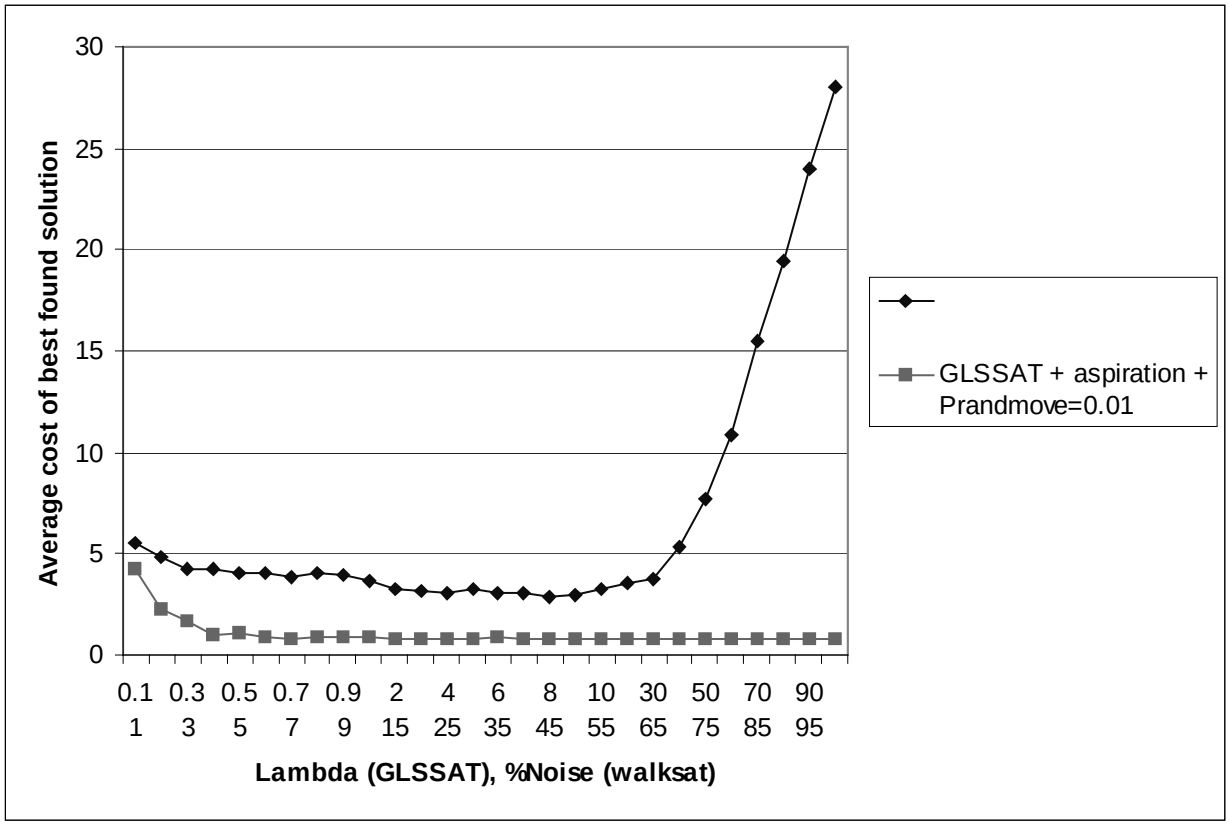
In Figure 5-2, we show the performance of the variants of GLSSAT with random walk (with Pr(randwalk) = 0.1) and aspiration moves. Again, there is little difference in the performance of the GLSSAT variants, although GLSSAT with aspiration again does slightly better for λ between 0.8 and 5, and the GLSSAT variants with random walk do slightly better when λ is 0.1. It appears that the random walk variants (at least

with Pr(randwalk) = 0.1) perform worse than GLSSAT or GLSSAT with just aspiration, except for λ values less than 0.3.



5.3.1.2 Comparing GLSSAT with walksat

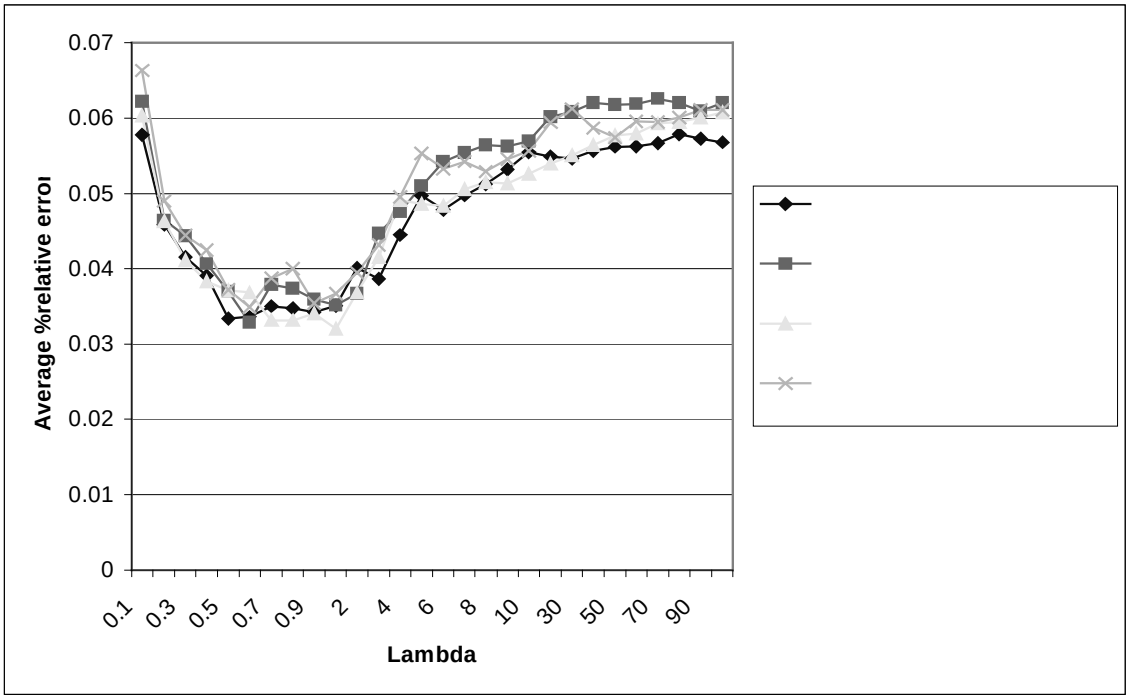
Figure 5-4 shows the performance of walksat over a range of parameter settings for noise. We can see that walksat performs quite well over a range of settings from 0 to 0.65, before the performance deteriorates rapidly, as the noise parameter is increased further, although the performance is still much worse than for any of the GLSSAT variants over the majority of parameter settings.



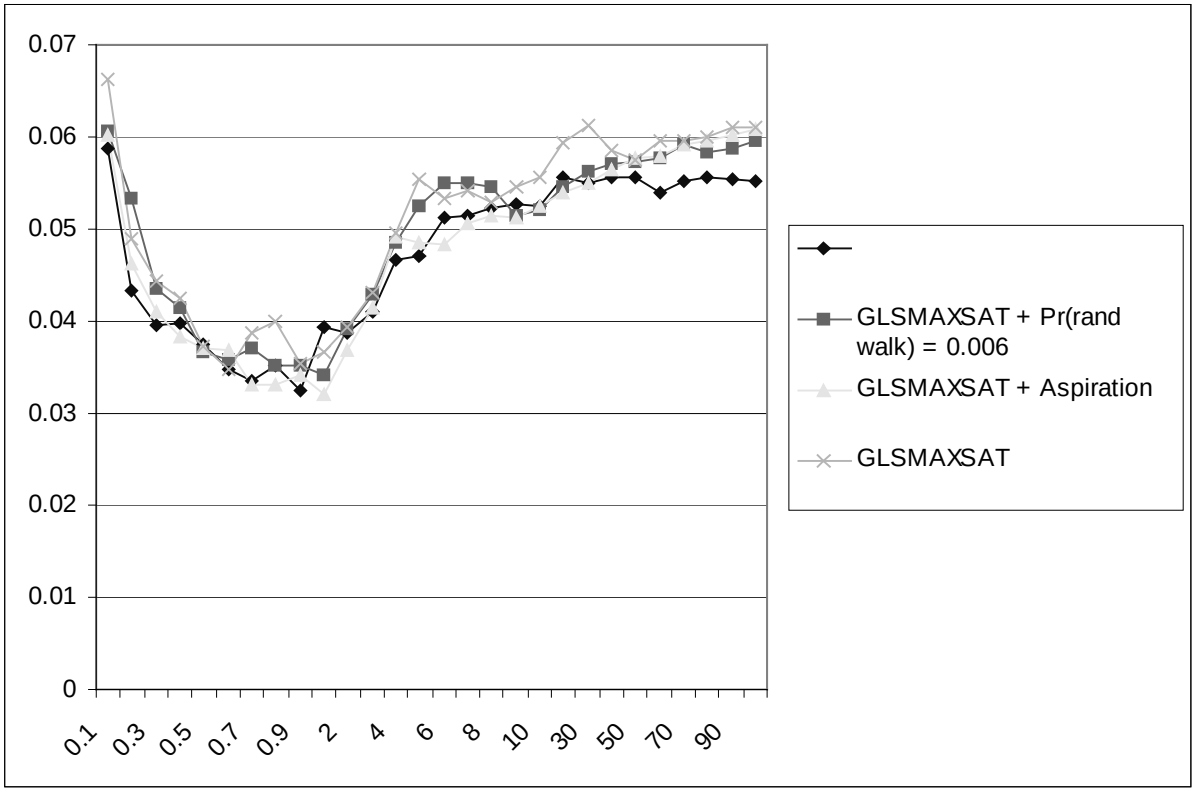
5.3.2 MAX-SAT Results

5.3.2.1 Comparing GLSMAXSAT variants

Figure 5-5 shows the performance of GLSMAXSAT with all combinations of the random moves and aspiration extensions. It can be seen that the variants with aspiration moves perform slightly better than the other variants of GLSMAXSAT, with the GLSMAXSAT variant with random moves and without aspiration performing the worst. GLSMAXSAT with aspiration moves and random moves performs better over most of the λ settings (except with λ settings of {0.6, 2}) than GLSMAXSAT with just random moves. From the figure, it is clear that the GLSMAXSAT variants with aspiration moves perform the best overall.



GLSMAXSAT with random walk and aspiration moves performs better than GLSMAXSAT with just random walk for most λ settings (all except when λ is in the set {0.5,2,9,10,11}), and also performs better than GLSMAXSAT with aspiration moves in some cases. GLSMAXSAT with no extensions performs the worst of all the variants tested. Clearly GLSMAXSAT with both random walk and aspiration moves peforms the best overall, due to its superiority over GLSMAXSAT with just aspiration moves for higher values of λ.



variants. GLSMAXSAT with just aspiration moves performs roughly the same as GLSMAXSAT with both random penalty walk and aspiration moves, except when λ takes any of the values {70, 80, 90, 100} when GLSMAXSAT with both extensions performs better. For this reason it appears that GLSMAXSAT with both random penalty walk and aspiration moves is superior to GLSMAXSAT with just aspiration moves, although there is little difference for λ settings less than 70.

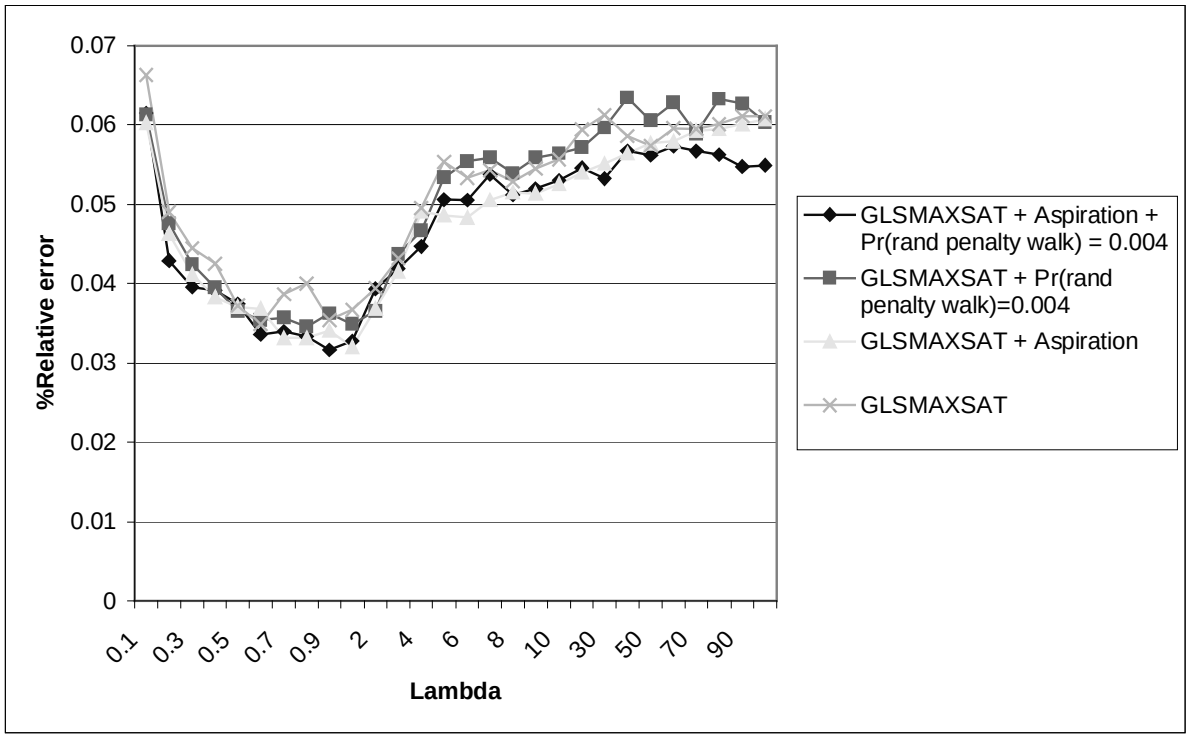
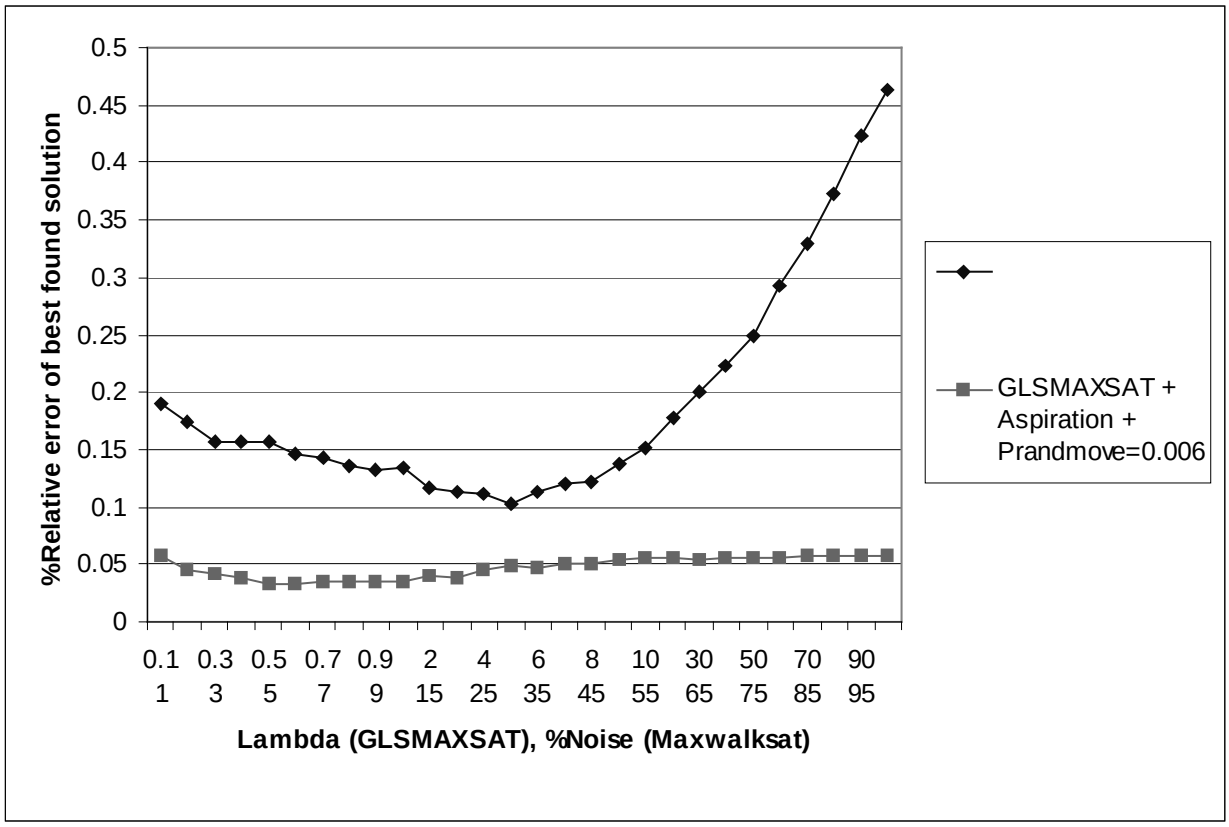


Figure 5-7: Performance of GLSMAXSAT with random penalty walk, with and without aspiration moves

5.3.2.2 Comparing GLSMAXSAT with MaxWalksat

In Figure 5-8, we show the performance of MaxWalksat over a large number of settings for its noise parameter, with its *best* performance being 0.1% relative error, as compared with a *worst* performance of a GLSMAXSAT variant being just under 0.07%. MaxWalksat's worst performance is just over 0.45% relative error.

MaxWalksat %relative error varies by 0.38% over all parameter settings, as compared with GLSMAXSAT's having a variation of just under 0.04% over all λ settings.



with the gap increasing as λ increases. GLSQAP with both aspiration moves and random moves performs well over all values of λ, although it performs worse than GLSQAP with just aspiration moves for larger values of λ (>1). The addition of aspiration moves to the GLSQAP scheme with random moves, seems to produce a better performance than one would expect, when λ is 0.5 and 0.6, although we have no explanation of why this should be. We suspect it may just be a feature of the set of test problems we used.

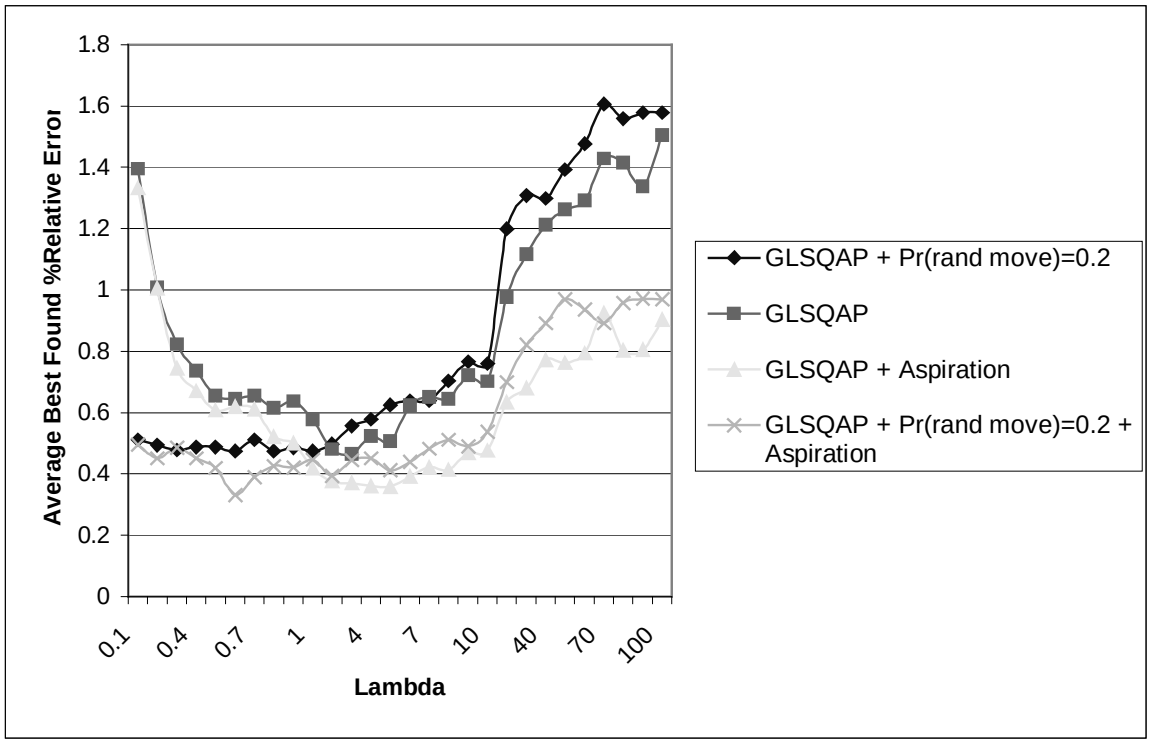


Figure 5-9: Performance of GLSQAP with and without random moves and aspiration moves

5.3.3.2 Comparing GLSQAP with Robust Tabu Search

Figure 5-10 and Figure 5-11 show the performance of Robust Tabu Search on the same QAP problems and under the same conditions as were used for the experiments

for GLSQAP, varying the u and t parameters respectively. In Figure 5-10, we can see that the

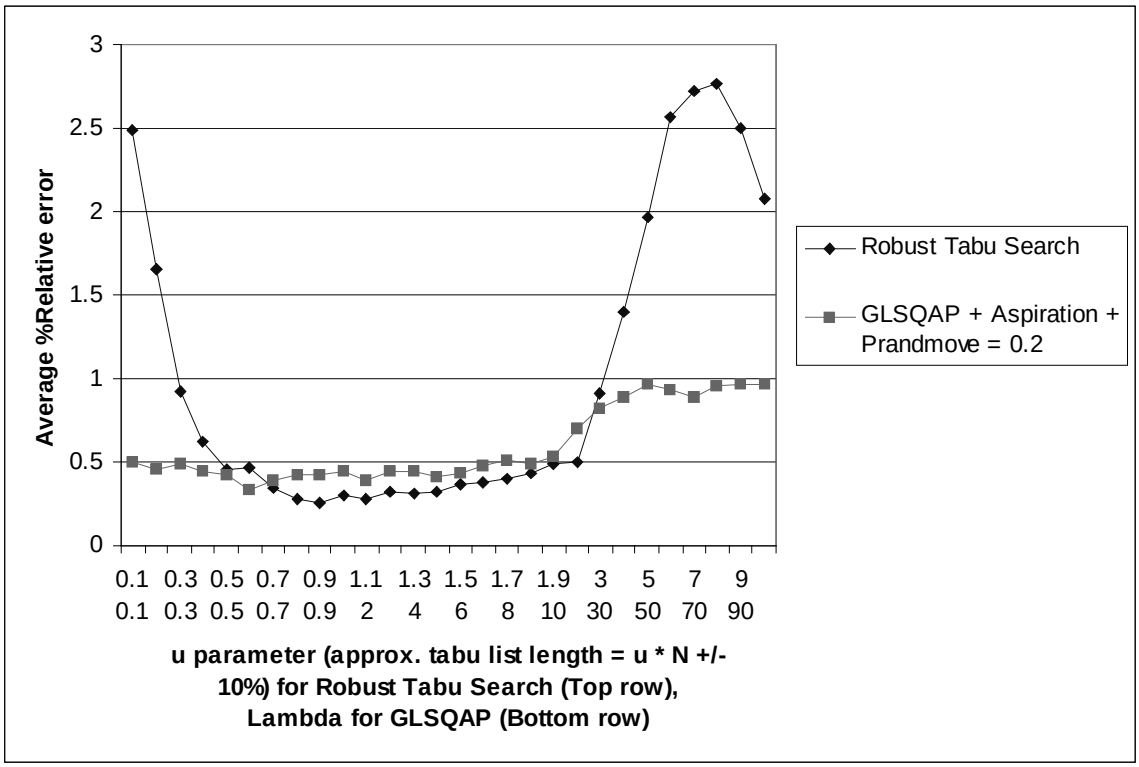
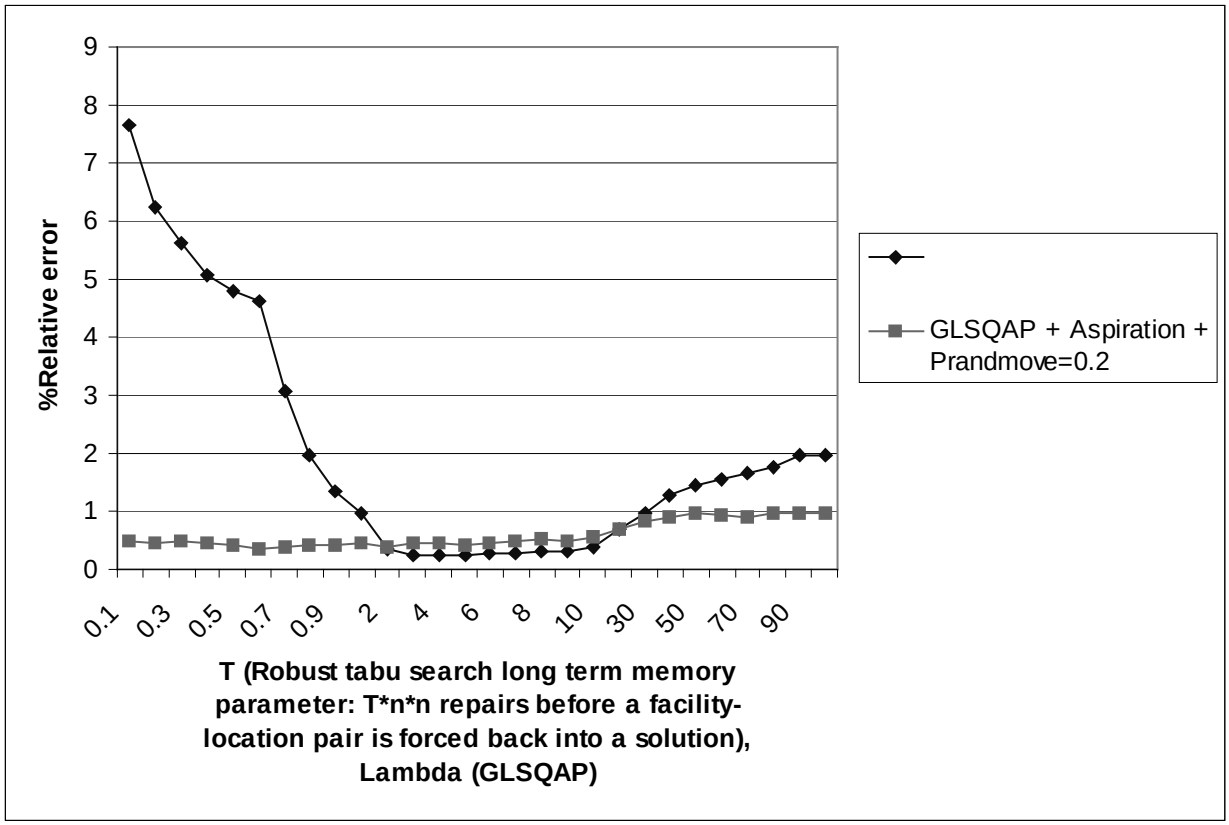


Figure 5-10: Performance of Robust Tabu Search, varying the u parameter (note: x-axis scales represent *different parameters* for each algorithm)

performance of robust tabu search varies between a relative error of just over 0.25 to just under 2.77, when we vary the u parameter. From the graph, we can see that any value for u between about 0.5**n* and 2.0**n* seems to give an average relative error of less than 0.5%, although performance rapidly deteriorates outside these bounds. In Figure 5-11, we can see the effect of varying the parameter t, which varies the number of repairs (given by tN^2) a facility-location assignment may be absent from any solutions, before it is forced into one, by robust tabu search's long term memory mechanism. When t is less than 2, performance becomes worse with decreasing t. When t is between 2 and 10, the performance of robust tabu search remains about the same. When t is greater than 10 performance gradually degrades with increasing t, up

to a relative error of 2%. We have no explanation of why this phenomenon occurs, although it is probably just a feature of the particular set of problems we used.



improvement over the performance of the basic GLSSAT. The performance of GLSQAP for low λ settings is improved when random moves are used, whilst for higher λ settings aspiration moves improve performance of GLSQAP over the basic GLSQAP. Combining both extensions produces both these positive effects, although there is a slight decrease in the performance over GLSQAP with just aspiration moves for higher λ values (the performance is still better than the basic GLSQAP with no extensions for these values, though).

5.4.2 Extended GLS verses state of the art algorithms

Comparing GLS with the other algorithms, we can see that GLSSAT performance is generally much better than walksat over a range of parameter settings. Comparing the performance of both algorithms with the best found parameter settings, the performance of GLSSAT (on average about 0.6 unsatisfied clauses per best found solution) is much better than walksat (about 3 unsatisfied clauses per best found solution). GLSMAXSAT performance was better than maxwalksat over all parameter settings, showing how good GLS is at solving problems involving soft constraints. GLSQAP's performance with aspiration moves and random moves is about the same as robust tabu search, although the performance varies much less (only 0.63% compared with 2.51% for the u parameter and 7.43% for the t parameter of Robust Tabu Search) with different λ settings than robust tabu search does with the u and t parameters (see Figure 5-10 & Figure 5-11). However, robust tabu search slightly outperforms GLSQAP with aspiration and random moves by 0.098%, when using the best-found parameter settings (u = 1, t = 5 for robust tabu search ; λ = 0.6, $P_{randmove}$ = 0.2 for GLSQAP) for both algorithms.

It should be noted though, that perhaps some method of cross-calibration of the algorithms is required to make the comparisons of performance over parameter settings more meaningful. Perhaps, this could be achieved by using the average entropy to find upper and lower bounds for each parameter of the algorithms, and then comparing performance over those parameter settings.

5.5 Discussion

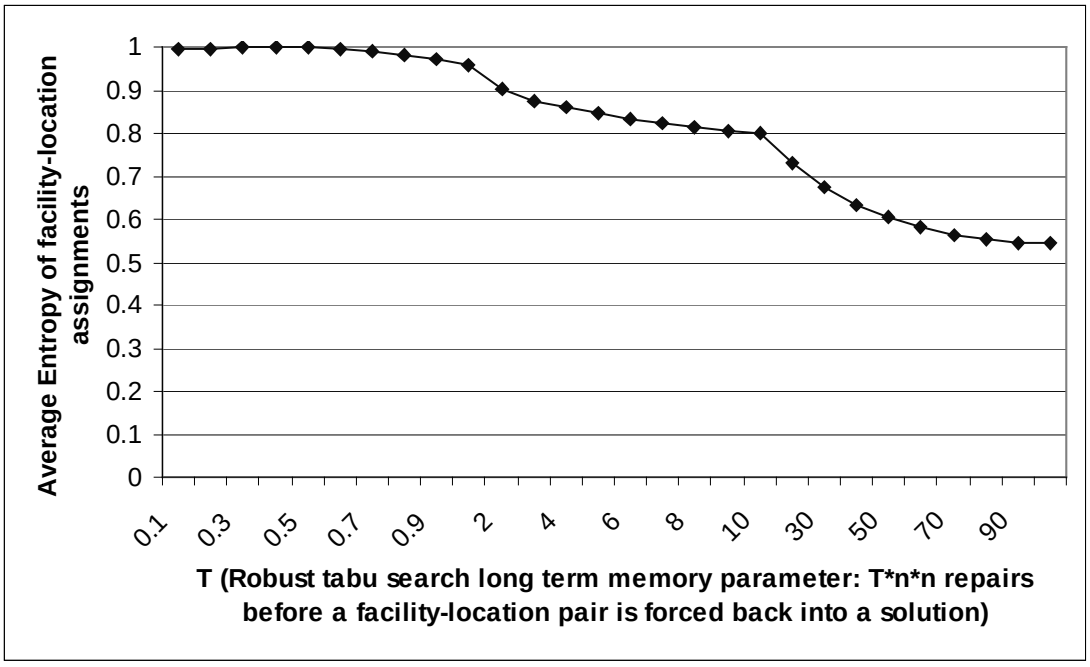
5.5.1 GLS and aspiration moves and random moves performance gap

The addition of aspiration moves to GLS decreases the relative error of algorithms with random moves (and also those with random walk and random penalty walk), more than those without for some settings of λ. This may be because it prevents random moves from being made when a new best-found solution exists in the current neighbourhood, forcing the search instead to make an aspiration move.

5.5.2 Robust Tabu Search long term memory compared to random moves

Another interesting feature of our comparison experiments was the effect that the t parameter (controlling how long a facility-location pair could be left out of any visited solutions) had on robust tabu search performance. Surprisingly this seemed to have a larger effect on its performance than the u parameter, which controls the tabu list length. This seems to indicate that this "long-term memory" mechanism for introducing location-facility pairs into the current solution, which have not been present for some time, improves the performance of RTS. We conjecture that random moves may also produce a similar effect on GLS, albeit randomly, rather than by forcing a particular facility-location pair into a solution (as in RTS), thus allowing the algorithm to visit areas of the search space which would otherwise not be visited.

Figure 5-12 shows that as the t parameter of robust tabu search ($t*n$ = number of repairs between when a facility-location pair was last present in a solution and when it will be forced into a solution again) is increased, the entropy (a strong indicator of the diversity of the search) decreases. Hence, this element of robust tabu search appears to help it increase the entropy (and therefore the diversity) of the search, allowing this mechanism to take effect more frequently produces higher entropy. In Chapter 4, we have seen how the entropy results of GLSQAP with random moves are also higher for GLSQAP without random moves. This partly substantiates the conjecture that random moves and robust tabu search's long term memory scheme do a similar job when solving the QAP: help diversify the search.



be implemented would be to have an additional set of "negative" penalty terms in the augmented objective function for helping to encourage facility-location pairs (or other types of feature, e.g. edges between cities in the TSP) which had not been present in a solution for a number of repairs to be reintroduced into a solution. These "incentive" penalties would have to be present for a fixed number of iterations after the reintroduction of the feature, to ensure the search had a chance to adapt the current solution to the new solution feature.

5.7 Conclusion

In this chapter, we have presented the results of combining aspiration moves and random moves with GLS, on the SAT, the weighted MAX-SAT and the QAP problems, in comparison to GLS without any extensions, and GLS with only one of aspiration moves, random moves, random walk or random penalty walk (the latter two only for the SAT and MAX-SAT problems). We have shown that GLS with random moves and aspiration moves gives either improved or comparable performance to GLS with either one of the extensions or without either extension. For the QAP problem, the Extended Guided Local Search (GLS with aspiration moves and random moves) performs better for both low λ settings (due mainly to the random move component) and high λ settings (due mainly to the aspiration moves). For the weighted MAX-SAT problem, we have shown that the Extended Guided Local Search (with either of random moves, random walk or random penalty walk extensions and aspiration moves) performs better for many parameter settings (and comparably for the rest) than the basic Guided Local Search or Guided Local Search with any one of the extensions only (although Guided Local Search with just aspiration moves performance is the closest of the other variants tried). For the SAT problem, the Extended Guided Local Search (again with either of random moves,

random walk or random penalty walk extensions and aspiration moves), performs comparably to basic Guided Local Search or Guided Local Search with any one extension only, although Guided Local Search with just aspiration moves appears to have a slight edge in terms of overall performance.

Finally, we have compared the extended GLS with other state of the art local search algorithms for each problem, over a range of parameter settings for both algorithms. We have shown that extended GLSSAT and extended GLSMAXSAT have much better performance for the SAT and MAX-SAT problems than Walksat and MaxWalksat. We have shown that extended GLSQAP has comparable performance to robust tabu search for the QAP, when compared over a range of parameter settings, whilst the performance of extended GLSQAP varies much less, over a range of λ settings, than robust tabu search does over its main parameter settings.

In summary Extended Guided Local Search (EGLS) performs better over a range of parameter settings for some problem types and performs comparably for other types to standard Guided Local Search. In addition to this, EGLS comfortably outperforms the famous Walksat and MaxWalksat algorithms for the SAT and weighted MAX-SAT problems, whilst performing comparably and in many cases better than Robust Tabu Search (RTS) over a range of parameter settings for the QAP.

6 Conclusion and Further work

In this chapter, we outline the contributions we have made and directions for further work.

6.1 Contributions

Our main contribution is in better understanding of search performance and in extending Guided Local Search. We elaborate on this, in the following sections.

6.1.1 Better understanding of search performance

In chapter 1, we introduced the concept of search monitors to help understand what effect each extension is having on the search. By doing this, we can better evaluate if an extension works as we expect, or if in fact something different is happening. This helps to remove the ad hoc trial and error testing of meta-heuristics, which has become common in the literature. In this way, as well as proposing extensions to GLS, we have also gained some understanding into why each of the extensions works for each problem type we tested.

6.1.2 Enhancing GLS with aspiration moves

In chapter 3 we have shown how the improved-best aspiration criterion can be used to extend Guided Local Search. We show this can be done, by simply examining the original objective function to check whether a new better than previously found solution exists in the current neighbourhood, before following the standard GLS scheme.

We have then shown how improved-best aspiration moves can on average improve the performance of Guided Local Search, for some parameter settings and problems.

Finally, we concluded that the improved-best aspiration criterion works mainly because of when it makes the overriding aspiration moves - namely, when a new better-than-previous found solution exists in the local search neighbourhood, meaning that GLS no longer misses these very important solutions. We have backed this up by performing a control experiment, involving a modified GLS, which (with a certain probability) performs local search according to the original objective function, otherwise it minimises the augmented objective function. We show that this variant of GLS does not perform as well as GLS with aspiration moves, and that the success of aspiration moves is not simply connected with every so often ignoring the penalty terms in the augmented objective function. Therefore it must be to do with *when* the penalty terms are ignored and an aspiration move made.

6.1.3 Enhancing GLS with random moves

In chapter 4, we show how every so often (according to some probability) making a move at random from the local search neighbourhood (or a subset of) may have some beneficial effects, when the λ parameter of GLS is set to a low value. We have shown that this is particularly the case for the QAP, where many local minima exist and that it is not so useful for problems like the SAT and weighted MAX-SAT where the local search landscape consists mainly of long plateaus. We have concluded that the reason for this is that random moves are unlikely to be as efficient at searching long plateaus as penalties, as they are not as systematic as penalties. This is because random moves may actually revisit already visited solutions. On the other hand, if penalties are used to search plateaus, revisiting of solutions is less likely to occur and moves to worse solutions are unlikely to occur until the plateau has been fairly thoroughly searched. We have backed this theory up with empirical evidence, which shows that random

moves actually lower the average entropy (strongly related to how diverse the search is) of solution labels, in comparison to GLS without random moves.

6.1.4 Reducing the sensitivity of performance to parameter settings

In chapter 5, we have shown how aspiration moves and random moves can be used together, improving over the performance of all other variants, over a range of parameter settings, in the QAP. We have then shown the extended GLS performs comparably or better than other state-of-the-art local search algorithms, over a wide range of parameter settings. Since GLS has only one major parameter λ, and our extensions have reduced the sensitivity of performance to this, then we believe this should make our extended GLS easier to apply to future problems.

6.2 Further work

6.2.1 A more advanced aspiration criterion for GLS

We would have like to have experimented with a more advanced aspiration criterion to see if we could improve over the basic improved-best aspiration criterion for GLS. This would allow the penalties to be ignored if a solution existed, such that it was of better quality (lower cost) than the worst of the best Q solutions visited so far. Aspiration moves to the current Q best found solutions would not be allowed, thus ensuring that this scheme will not cause solutions to be revisited. This would allow us to vary the amount of aspiration moves. Thus, we would be able to see if allowing more aspiration moves had any effect on the quality of solutions and if it would be possible to produce even better results. Hopefully, this would lead to GLS exploring higher quality basins and plateaus in the search landscape, which might otherwise have been ignored due to penalties imposed earlier on in the search.

6.2.2 A more advanced random move scheme

We would also have liked to experiment with a more elaborate random move scheme. The first way in which we might improve the scheme would be to only allow random moves to be made once GLS was in a local minimum as an alternative to penalising features in that local minimum. In addition to this, the way in which a random move is made could be extended by allowing a sequence of random moves from the neighbourhood to be made, rather than just one individual random move, thus having a greater effect on the solution. The number of random moves could be fixed according to some parameter setting, be varied randomly, or varied according to the quality of the current local minimum or learnt during the search in some way. By using such a scheme, where larger jumps may be made, this might also alleviate problems with poor random starting points, by allow a partial restarting strategy.

6.2.3 Long term memory using penalty incentives

Another aspect is the need to make sure the whole of the search space is explored as evenly as possible for good quality regions. To this end, it may also be useful to look at adding long term memory diversification strategies, such as are used by robust tabu search [89], whereby solution attributes which have not been present for more than a specified number of iterations are forced into the current solution. A softer version of this could be implemented in GLS by the addition of incentives (negative penalties) to the augmented objective function (present before, and then after the desired attribute has been present in solutions for a number fixed number of repairs), which encourage these attributes to be re-introduced into solutions. Because the incentives would not actually force the attributes into solutions, but only "encourage" them with the right amount of incentive, it might be that the local search algorithm would only place these attributes in solutions when they were relatively advantageous to the current

area of the search space. This might increase the likelihood of this resulting in higher quality regions of the search space being discovered by such a mechanism.

7 Bibliography

1. Amin, A.: Simulated Jumping. In Annals of Operations Research 86, pages 23-38, 1999.
2. Angel, E. & Zissimopoulos, V.: On the hardness of the Quadratic Assignment Problem with meta-heuristics. Technical Report, Laboratoire de Researche en Informatique, Universite Paris Sud, France, 1997.
3. Angel, E. & Zissimopoulos, V.: On the landscape ruggedness of the Quadratic Assignment Problem. Technical Report 98-02, LIPN, Institut Galilee, Universitie de Paris-Nord, 1998.
4. Battiti, R. & Tecchiolli, G.: The Reactive Tabu Search. In ORSA Journal on Computing, 6(2), pages 126-140, 1994.
5. Battiti, R. & Tecchiolli, G.: Local search with memory: Benchmarking RTS. In Operations Research Spektrum, 17(2/3), pages 67-86, 1995.
6. Bentley, J.L. Fast Algorithms for Geometric Travelling Salesman Problems. ORSA Journal on Computing, 4(4), pages 387-411, 1992.
7. Bishop, C.M.: Neural Networks for Pattern Recognition. Clarendon Press 1995.
8. Burkard, R.E., Karisch, S.E., & Rendl, F.: QAPLIB - A Quadratic Assignment Problem Library. In Journal of Global Optimization 10, pages 391-403, 1997.
9. Boyce, J.F., Dimitropoulos, C.H.D., vom Scheidt, G., & Taylor, J.G.: GENET and Tabu Search for Combinatorial Optimization Problems. In World Congress on Neural Networks, Washington D.C., 1995.
10. Cha, B. & Iwama, K.: Performance tests of local search algorithms using new types of random cnf formulas. In Proceedings of IJCAI'95, Volume 1, pages 304-310, 1995.

11. Chang, Y.-J. and Wah, B.W.: Lagrangian Techniques for Solving a Class of Zero-One Integer Linear Programs. In Proceedings International Conference on Computer Software and Applications, IEEE, pages 156-161, 1995.
12. Choi, K.M.F., Lee, J.H.M. & Stuckey, P.J.: 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, 1988.
13. Clarke, G.M. & Cooke, D.: A basic course in statistics - 3rd ed. Edward Arnold, a division of Hodder and Stoughton Limited, 1992.
14. Cung, V., Mautor, T., Michelon, P. & Tavares, A.: A Scatter Search Based Approach for the Quadratic Assignment Problem. In Proceedings of IEEE International Conference on Evolutionary Computation and Evolutionary Programming, Indianapolis, United States of America, pages 161-170, 1997.
15. Dammeyer, F. & Voβ, S.: Dynamic tabu list management using the reverse elimination method. In Annals of Operations Research 41, pages 31-46, 1993.
16. Davenport, A.J., Tsang, E.P.K., Wang, C.J. and Zhu, K.: GENET: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. In Proceedings of AAAI-94, Vol. 1, pages 325-330, 1994.
17. Davenport, A.: A comparison of complete and incomplete algorithms in the easy and hard regions. In Proceedings, Workshop on Studying and Solving Really Hard Problems, First International Conference on Principles and Practice of Constraint Programming, pages 43-51, 1995.
18. Davenport A. & Tsang E.P.K.: Solving constraint satisfaction sequencing problems by iterative repair. In Proceeding, 14th UK Planning and Scheduling Special Interest Group Workshop, Colchester, November, 1995.

19. Davenport, A.J. & Tsang, E.P.K., Solving constraint satisfaction sequencing problems by iterative repair. In Proceedings of The First International Conference on The Practical Application of Constraint Technologies and Logic Programming (PACLP), London, pages 345-357, April 1999.
20. Davenport, A.: Extensions and Evaluation of GENET in Constraint Satisfaction, Ph.D. thesis. Department of Computer Science, University of Essex, 1997.
21. Davis, M. and Putnam, H.: A computing procedure for quantification theory. Journal of the ACM 7, pages 201-215, 1960.
22. Dorigo, M., Maniezzo, V. and Colorni, A.: The Ant System: Optimization by a colony of cooperating agnets. In IEEE Transactions on Systems, Man, and Cybernetics - Part B, Vol. 26, No. 1, pages 1-13, 1996.
23. Fox, B.L.: Integrating and accelerating tabu search, simulated annealing, and genetic algorithms. In Annals of Operations Research 41, pages 47-67, 1993.
24. Fleurent, C., Ferland, J.A.: 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, 1994.
25. Frank, J.: Weighting for Godot: Learning heuristics for GSAT. In Proceedings of AAAI'96, pages 338-343, 1996.
26. Frank, J.: Learning Short-Term Weights for GSAT. In Proceedings IJCAI '97, Volume 1, pages 384-389, 1997.
27. Frank, J., Cheeseman, P. and Stutz, J.: When Gravity Fails: Local Search Topology. In Journal of Artificial Intelligence Research 7, pages 249-291, 1997.
28. Freuder, E.C. & Wallace, R.J.: Partial constraint satisfaction, Artificial Intelligence, Vol.58, Nos.1-3, (Special Volume on Constraint Based Reasoning), pages 21-70, 1992.

29. Gambardella, L.M.,Taillard,E.D. & Dorigo,M.: Ant Colonies for the QAP. Accepted for publication in the Journal of the Operational Research Society, 50, pages 167-176, 1999.
30. Garey, M.R. & Johnson, D.S.: Computers and intractability, W.H. Freeman and Company, 1979.
31. Gent, I.P., van Maaren, H. & Walsh, T.: SAT2000, Highlights of satisfiability research in the year 2000, Frontiers in Artificial Intelligence and Applications, IOS Press, 2000.
32. Gent, I. & Walsh, T.: Unsatisfied Variables in Local Search. In Hybrid Problems, Hybrid Solutions, ed. J. Hallam, IOS Press, Amsterdam (Proceedings of AISB-95), pages 73-85, 1995.
33. Gent, I. & Walsh, T.: Towards an Understanding of Hill-climbing Procedures for SAT. In Proceedings of AAAI-93, pages 28-33, 1993.
34. Ginsberg, M. & McAllester, D.: GSAT and Dynamic Backtracking. In Fourth Conference on Principles of Knowledge Representation and Reasoning (KR-94), pages 226-237, 1994.
35. Gu, J.: Local search for Satisfiability (SAT) Problem. In IEEE Transactions on Systems, Man and Cybernetics, Vol. 23, No. 4, pages 1108-1129, 1993.
36. Gu, J.: Global Optimization for Satisfiability (SAT) Problem. In IEEE Transactions on Knowledge and Data Engineering, Vol. 6, No. 3, pages 361-381, 1994.
37. Glover, F.: Tabu search Part I. Operations Research Society of America (ORSA), Journal on Computing, Vol. 1, pages 109-206, 1989.
38. Glover, F.: Tabu search Part II. Operations Research Society of America (ORSA), Journal on Computing, Vol. 2, pages 4-32, 1990.

39. Glover, F., Taillard, E. & de Werra, D.: A user's guide to tabu search. In Annals of Operations Research 41, pages 3-28, 1993.
40. Glover, F. & Laguna, M.: Tabu Search. Kluwer Academic Publishers, 1997.
41. Holland, J.H.: 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.
42. Hooker, J.N.: Testing Heuristics: We Have It All Wrong. In Journal of Heuristics, Volume 1, Number 1, pages 33-42, Fall 1995.
43. Jiang Y., Kautz H., and Selman B.: 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.
44. Johnson, D.S. & Trick, M.A. (eds.), Second DIMACS Implementation Challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1993.
45. Kask, K & Dechter, R.: GSAT and Local Consistency. In Proceedings of IJCAI'95, pages 616-622, 1995.
46. Kautz, H., McAllester, D. & Selman, B.: Exploiting Variable Dependency in Local Search. Abstract appears in Abstracts of the Poster Sessions of IJCAI-97, Nagoya, Japan, 1997.
47. Kilby, P., Prosser, P. & Shaw, P.: Guided Local Search for the Vehicle routing Problem, In Proceedings of the 2nd International Conference on Metaheuristics, pages 21-24, 1997.
48. Kirkpatrick, S., Gelatt, C.D. & Vecchi, M.P.: Optimisation by simulated annealing. In Science 220 (1983) 671-680.
49. Koza, J.R.: Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press, 1994.

50. Lau, T.L.: Guided Genetic Algorithm, PhD Thesis, Department of Computer Science, University of Essex 1999.
51. Lau, T.L. and Tsang, E. P. K.: Solving the generalized assignment problem with the guided genetic algorithm. In Proceedings of 10th IEEE International Conference on Tools for Artificial Intelligence, pages 336-343, November 1998.
52. Lau, T.L. and Tsang, E. P. K.: Guided Genetic algorithm and its application to the large processor configuration problem. In Proceedings of 10th IEEE International Conference on Tools for Artificial Intelligence, pages 320-327, November 1998.
53. Lau, T.L. & Tsang, E.P.K.: Solving the radio link frequency assignment problem with the guided genetic algorithm. In Proceedings of NATO Symposium on Radio Length Frequency Assignment, Sharing and Conservation Systems (Aerospace), Aalborg, Demark, Paper 14b, October 1998.
54. Lau, T.L. & Tsang, E.P.K.: Guided genetic algorithm and its application to radio link frequency assignment problems. In Journal of Constraints, Vol. 6, No. 4, pages 374-398, 2001.
55. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. & Shmoys, D.B. (ed.): The travelling salesman problem - a guided tour of combinatorial optimization, John Wiley & Sons, 1985.
56. Lee, J.H.M., Leung, H.F. & Won, H.W.: 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, November 1995.
57. Lee, J.H.M., Leung, H.F. & Won, H.W.: 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, August 1996.

58. Lee, J.H.M., Leung, H.F. & Won, H.W.: 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, July, 1998.

59. McAllester, D., Selman, B. & Kautz, H.: Evidence for Invariants in Local Search. In Proceedings AAAI-97, Providence, RI, pages 321-326, 1997.

60. Merz, P. and Freisleben, B.: 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, 1999.

61. Merz, P. and Freisleben, B.: Fitness Landscapes and Memetic Algorithm Design. (Chapter 3) In New Ideas in Optimization (D.Corne, M.Dorigo, F.Glover eds.), McGraw-Hill, pages 244-260, 1999.

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

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

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

65. Mladenovic,N. & Hansen, P.: Variable Neighborhood Search. In Computers in Operations Research, Vol. 24, No. 11, pages 1097-1100, 1997.

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

67. Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms, C3P Report 826, Caltech Concurrent Computation Program, Caltech, California, USA, 1989.

68. Moscato, P.: 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, 1993.

69. Pardalos,P.M., Rendl,F., Wolkowicz, H.: The Quadratic Assignment Problem: A Survey of Recent Developments. In P. Pardalos and H. Wolkowicz, editors, Quadratic Assignment and Related Problems, volume 16, pages 1-42, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1994.

70. Reeves, C.R.: Fitness landscapes and Evolutionary Algorithms. EA99 Invited talk, 1999.

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

72. Reeves, C.R. & Yamada, T.: Genetic Algorithms, Path Relinking and the Flowshop Sequencing Problem. In Evolutionary Computation, 6.1, pages 45-60, 1998.

73. Reeves, C.R.: Predictive Measures for Problem Difficulty. In Proceedings of 1999 Congress of Evolutionary Computation, IEEE Press, Piscataway, NJ, pages 736-743, 1999.

74. Reeves, C.R.: Hybrid genetic algorithms for bin-packing and related problems. In Annals of Operations Research 63, pages 371-396, 1996.

75. Resende M. G. C., Pitsoulis L. S., and Pardalos P.M.: Approximate Solution of Weighted MAX-SAT Problems using GRASP. In DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 35, pages 393-405, 1997.

76. Richards, T. & Richards, B.: Non-systematic search and learning: an empirical study. In Maher, M. & Puget, J-F. (ed.), Proceedings, 4th International Conference on Principles and Practice of Constraint Programming, Pisa, Italy, Lecture Notes in Computer Science 1520, Springer Verlag, pages 370-384, October 1998.
77. Selman, B. & Kautz, H.: An Empirical Study of Greedy Local Search for Satisfiability Testing. In Proceedings AAAI-93, pages 46-51, 1993.
78. Selman, B. & Kautz, H.: Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems. In Proceedings IJCAI-93, pages 290-295, 1993.
79. Selman, B., Kautz, H. & Cohen, B.: Noise Strategies for Improving Local Search. In Proceedings AAAI-94, pages 337-343, 1994.
80. Selman, B., Kautz H. & McAllester, D.: Ten Challenges in Propositional Reasoning and Search. In Proceedings of the Fifteenth International Conference on Artificial Intelligence (IJCAI-97), NAGOYA, Aichi, Japan, pages 50-54, 1997.
81. Selman, B., Levesque, H. & Mitchell, D.: A New Method for Solving Hard Satisfiability Problems. In Proceedings AAAI-92, pages 440-446, 1992.
82. Shang, Y.: Global Search Methods for Solving Nonlinear Optimization Problems. Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, August, 1997.
83. Shang, Y. & Wah, B. W.: 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, November, 1998.
84. Shang, Y. & Wah, B.W.: A Discrete Lagrangian-Based Global-Search Method for Solving Satisfiability Problems. In Journal of Global Optimization, Kluwer Academic Publishers, vol. 12, no. 1, pages 61-99, 1998.

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

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

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

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

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

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

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

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

93. Thonemann, U.W., Bolte, A.: An Improved Simulated Annealing Algorithm for the Quadratic Assignment Problem. Working paper, School of Business, Department of Production and Operations Research, University of Paderborn, Germany, 1994.

94. Tsang, E.P.K.: Foundations of Constraint Satisfaction. Academic Press, 1993.

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

96. Tsang, E.P.K., Voudouris, C.: Fast Local Search and Guided Local Search and their application to British Telecom's Workforce Scheduling Problem. In Operations Research Letters, Elsevier Science Publishers, Amsterdam, Vol. 20, No. 3, pages 119-127, March 1997.

97. Tsang, E.P.K., Wang, C.J., Davenport, A., Voudouris, C., Lau,T.L.: 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, April 1999.

98. Voudouris, C. & Tsang, E.: The Tunneling Algorithm for Partial CSPs and Combinatorial Optimization Problems. Technical Report CSM-213, University of Essex, Colchester, UK, September 1994.

99. Voudouris, C. & Tsang, E.: Guided Local Search. Technical Report CSM-247, Department of Computer Science, University of Essex, Colchester, UK, August 1995.

100. Voudouris, C. & Tsang, E.: Function Optimization using Guided Local Search. Technical Report CSM-249. Department of Computer Science, University of Essex, Colchester, UK, September 1995.

101. Voudouris, C.: Guided Local Search – an illustrative example in function optimisation. In BT Technology Journal, Vol. 16, No. 3, pages 46-50, 1998.

102. Voudouris, C.: Guided Local Search for Combinatorial Optimisation Problems, Ph.D. thesis. Department of Computer Science, University of Essex, 1997.

103. Voudouris, C. and Tsang, E.P.K.: Guided local search and its application to the traveling salesman problem. In European Journal of Operational Research, Vol.113, Issue 2, pages 469-499, November 1998.

104. Voudouris, C. and Tsang, E.P.K.: Solving the Radio Link Frequency Assignment Problem using Guided Local Search. In Proceedings, NATO Symposium on Radio Length Frequency Assignment, Sharing and Conservation Systems (Aerospace), Aalborg, Denmark, Paper 14, October 1998.

105. Voudouris, C. & Tsang, E.P.K.: Guided local search joins the elite in discrete optimisation. In DIMACS Series in Discrete Mathematics and Theoretical Computer Science Volume 57, pages 29-39, 2001.

106. Wah, B.W. & Wu Z.: Discrete Lagrangian Methods for Designing Multiplierless Two-Channel PR-LP Filter Banks. In Journal of VLSI Signal Processing, Kluwer Academic Press, vol. 21, no. 2, pages 131-150, June 1999.

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

108. Wah, B.W., Shang, Y. & Wu, Z.: Discrete Lagrangian Methods for Optimizing the Design of Multiplierless QMF Banks. In IEEE Trans. on Circuits and Systems, Part II, vol. 46, no. 9, pages 1179-1191, September 1999.

109. Wah, B.W., Wang, T., Shang Y. & Zhe W.: Improving the Performance of Weighted Lagrange-Multiplier Methods for Nonlinear Constrained Optimization. In Proc. 9th International Conference on Tools with Artificial Intelligence IEEE, pages 224-231, November 1997.

110. Warners, J.P. and van Maaren, H.: A Two Phase Algorithm for Solving a Class of Hard Satisfiability Problems. Report SEN-R9802 (CWI). In Operations Research Letters 23, pages 81-88, 1999.

111. Wilhelm, M.R., Ward, T.L.: Solving Quadratic Assignment Problems by Simulated Annealing. In IIE Transaction 19(1), pages 107-119, 1987.

112. Woodruff, D.L.: Hashing vectors for tabu search. In Annals of Operations Research 41, pages 123-137, 1993.

113. Wu, Z.: 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, May 1998.

114. Wu, Z.: The Theory and Applications of Discrete Constrained Optimization using Lagrange Multipliers. Ph.D. Thesis, Dept. of Computer Science, University of Illinois, May 2001.

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

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

117. Xu, J., S. Y. Chiu and F. Glover: Fine-Tuning a Tabu Search Algorithm with Statistical Tests. Technical Report, University of Colorado at Boulder, 1996.

8 Appendix A: Details of problems used in experiments

In this section, we give details of the problems used for the experiments in Chapters 3,4 and 5.

8.1 SAT Problems

Table 3 below lists the 129 SAT problems used in the experiments described in this thesis, along with the number of variables and clauses in each problem.

Problem	#Vars	#Clauses	Problem	#Vars	#Clauses	Problem	#Vars	#Clauses
aim-100-1_6-yes1-1	100	160	aim-50-3_4-yes1-4	50	170	ii32e4	387	7106
aim-100-1_6-yes1-2	100	160	aim-50-6_0-yes1-1	50	300	ii32e5	522	11636
aim-100-1_6-yes1-3	100	160	aim-50-6_0-yes1-2	50	300	ii8a1	66	186
aim-100-1_6-yes1-4	100	160	aim-50-6_0-yes1-3	50	300	ii8a2	180	800
aim-100-2_0-yes1-1	100	200	aim-50-6_0-yes1-4	50	300	ii8a3	264	1552
aim-100-2_0-yes1-2	100	200	as10-yes	216	2780	ii8a4	396	2798
aim-100-2_0-yes1-3	100	200	as11-yes	112	1312	ii8b1	336	2068
aim-100-2_0-yes1-4	100	200	as12-yes	72	1012	ii8b2	576	4088
aim-100-3_4-yes1-1	100	340	as13-yes	232	2276	ii8b3	816	6108
aim-100-3_4-yes1-2	100	340	as14-yes	92	758	ii8b4	1068	8214
aim-100-3_4-yes1-3	100	340	as15-yes	232	3692	ii8c1	510	3065
aim-100-3_4-yes1-4	100	340	as2-yes	96	954	ii8c2	950	6689
aim-100-6_0-yes1-1	100	600	as3-yes	96	954	ii8d1	530	3207
aim-100-6_0-yes1-2	100	600	as4-yes	328	4176	ii8d2	930	6547
aim-100-6_0-yes1-3	100	600	as5-yes	1208	31124	ii8e1	520	3136
aim-100-6_0-yes1-4	100	600	as6-yes	184	2277	ii8e2	870	6121
aim-200-1_6-yes1-1	200	320	as7-yes	760	17896	jnh1	100	850
aim-200-1_6-yes1-2	200	320	as8-yes	84	974	jnh12	100	850
aim-200-1_6-yes1-3	200	320	ii16a1	1650	19368	jnh17	100	850
aim-200-1_6-yes1-4	200	320	ii16a2	1602	23281	jnh201	100	800
aim-200-2_0-yes1-1	200	400	ii16b1	1728	24792	jnh204	100	800
aim-200-2_0-yes1-2	200	400	ii16b2	1076	16121	jnh205	100	800
aim-200-2_0-yes1-3	200	400	ii16c1	1580	16467	jnh207	100	800
aim-200-2_0-yes1-4	200	400	ii16c2	924	13803	jnh209	100	800
aim-200-3_4-yes1-1	200	680	ii16d1	1230	15901	jnh210	100	800
aim-200-3_4-yes1-2	200	680	ii16d2	836	12461	jnh212	100	800
aim-200-3_4-yes1-3	200	680	ii16e1	1245	14766	jnh213	100	800
aim-200-3_4-yes1-4	200	680	ii16e2	532	7825	jnh217	100	800
aim-200-6_0-yes1-1	200	1200	ii32a1	459	9212	jnh218	100	800
aim-200-6_0-yes1-2	200	1200	ii32b1	228	1374	jnh220	100	800
aim-200-6_0-yes1-3	200	1200	ii32b2	261	2558	jnh301	100	900
aim-200-6_0-yes1-4	200	1200	ii32b3	348	5734	jnh7	100	850
aim-50-1_6-yes1-1	50	80	ii32b4	381	6918	par8-1-c	64	254
aim-50-1_6-yes1-2	50	80	ii32c1	225	1280	par8-2-c	68	270
aim-50-1_6-yes1-3	50	80	ii32c2	249	2182	par8-3-c	75	298
aim-50-1_6-yes1-4	50	80	ii32c3	279	3272	par8-4-c	67	266
aim-50-2_0-yes1-1	50	100	ii32c4	759	20862	par8-5-c	75	298
aim-50-2_0-yes1-2	50	100	ii32d1	332	2703	ssa7552-038	1501	3575
aim-50-2_0-yes1-3	50	100	ii32d2	404	5153	ssa7552-158	1363	3034
aim-50-2_0-yes1-4	50	100	ii32d3	824	19478	ssa7552-159	1363	3032
aim-50-3_4-yes1-1	50	170	ii32e1	222	1186	ssa7552-160	1391	3126
aim-50-3_4-yes1-2	50	170	ii32e2	267	2746	tm1-yes	2421	40723
aim-50-3_4-yes1-3	50	170	ii32e3	330	5020	tm2-yes	337	1888

Table 3: Details of SAT problems used in experiments

8.2 MAX-SAT Problems

Table 4 below lists the 44 MAX-SAT problems used in the experiments described in this thesis, along with the number of variables and clauses in each problem.

Problem	#Vars	#Clauses	Problem	#Vars	#Clauses	Problem	#Vars	#Clauses	Problem	#Vars	#Clauses
jnh1	100	850	jnh201	100	800	jnh215	100	800	jnh306	100	900
jnh10	100	850	jnh202	100	800	jnh216	100	800	jnh307	100	900
jnh11	100	850	jnh203	100	800	jnh217	100	800	jnh308	100	900
jnh12	100	850	jnh205	100	800	jnh218	100	800	jnh309	100	900
jnh13	100	850	jnh207	100	800	jnh219	100	800	jnh310	100	900
jnh14	100	850	jnh208	100	800	jnh220	100	800	jnh4	100	850
jnh15	100	850	jnh209	100	800	jnh301	100	900	jnh5	100	850
jnh16	100	850	jnh210	100	800	jnh302	100	900	jnh6	100	850
jnh17	100	850	jnh211	100	800	jnh303	100	900	jnh7	100	850
jnh18	100	850	jnh212	100	800	jnh304	100	900	jnh8	100	850
jnh19	100	850	jnh214	100	800	jnh305	100	900	jnh9	100	850

Table 4: Details MAX-SAT problems used in experiments

8.3 QAP Problems

Table 5 below lists the 94 QAP problems used in the experiments described in this thesis, along with the number of elements in the permutation for each problem.

Problem	Permutation Size	Problem	Permutation Size	Problem	Permutation Size
bur26a	26	esc32a	32	nug24	24
bur26b	26	esc32b	32	nug25	25
bur26c	26	esc32c	32	nug30	30
bur26d	26	esc32d	32	rou12	12
bur26e	26	esc32e	32	rou15	15
bur26f	26	esc32f	32	rou20	20
bur26g	26	esc32g	32	scr12	12
bur26h	26	esc32h	32	scr15	15
chr12a	12	had12	12	scr20	20
chr12b	12	had14	14	ste36a	36
chr12c	12	had16	16	ste36b	36
chr15a	15	had18	18	ste36c	36
chr15b	15	had20	20	tai10a	10
chr15c	15	kra30a	30	tai10b	10
chr18a	18	kra30b	30	tai12a	12
chr18b	18	lipa20a	20	tai12b	12
chr20a	20	lipa20b	20	tai15a	15
chr20b	20	lipa30a	30	tai15b	15
chr20c	20	lipa30b	30	tai17a	17
chr22a	22	lipa40a	40	tai20a	20
chr22b	22	lipa40b	40	tai20b	20
chr25a	25	nug12	12	tai25a	25
els19	19	nug14	14	tai25b	25
esc16a	16	nug15	15	tai30a	30
esc16b	16	nug16a	16	tai30b	30
esc16c	16	nug16b	16	tai35a	35
esc16d	16	nug17	17	tai35b	35
esc16e	16	nug18	18	tai40a	40
esc16g	16	nug20	20	tai40b	40
esc16h	16	nug21	21	tho30	30
esc16i	16	nug22	22	tho40	40
esc16j	16				

Table 5: Details of QAP problems used in experiments

9 Appendix B: Full Results

Appendix B is on the CD-ROM provided with this thesis. It contains excel spreadsheets of the results for all the experiments described in this thesis, as well as many other statistics from the experiments. It should be noted however, that some of these statistics may not be valid (for example, statistics about aspiration moves when no aspiration moves were made or statistics that were not recorded to save CPU time. These statistics usually have NA,NU, 0, or - as values in the results tables).

We group the sets of spreadsheets in 8 directories listed below:

- Aspiration_Results - contains spreadsheets with data on all the experiments involving GLS variants with the aspiration move extension,
- Basic_GLS_Results - contains spreadsheets with data on all the experiments involving just the basic GLS with no extensions,
- Combinations_Results - contains spreadsheets with data on all experiments involving GLS with both random moves (or random walk or random penalty walk) and aspiration moves,
- Cost_Plot_Results - contains spreadsheets with data on the costs of solutions visited during a run of GLS on examples of each problem type from the random moves chapter,
- Ignore_Penalties_Results - contains spreadsheets with data on all the control experiments from Chapter 3 involving GLS ignoring penalties,
- Other_Algorithms_Results - contains spreadsheets with data on the performance of Robust Tabu Search, walksat and maxwalksat,
- Random_Move_Results - contains spreadsheets with data on the performance of GLS with the random move, random walk and random penalty walk extensions.

- Significance_Tests - contains a spreadsheet with details of the sign tests performed on some of the figures in chapters 3 & 4.

Each spreadsheet (these can be read using Microsoft Excel) contains data in the form of multiple tables. The most common table is as follows (although the Cost_Plot_Results directory are not). The first table of each spreadsheet is usually a summary of the overall results in terms of performance over the parameter settings tried in that experiment. The next table down and all the rest of the tables give the average, standard deviation, minimum and maximum values out of 10 runs for different parameter settings (1 parameter setting per column) and problems (1 problem per row), with 1 table for each search monitor. See Table 6 for a cut-down example of such a table (showing the average fraction of runs per problem, where solutions satisfying all clauses were found by GLSSAT).

Averages for Sol_Found					
	lambda=0.1	lambda=0.2	lambda=0.3	lambda=0.4	lambda=0.5
aim-100-1_6-yes1-1	0.0	0.1	0.0	0.0	0.0
aim-100-1_6-yes1-2	0.0	0.0	0.0	0.0	0.0
aim-100-1_6-yes1-3	0.0	0.0	0.1	0.0	0.0
aim-100-1_6-yes1-4	0.0	0.0	0.0	0.0	0.0
aim-100-2_0-yes1-1	0.0	0.0	0.0	0.0	0.0
aim-100-2_0-yes1-2	0.0	0.0	0.0	0.0	0.0
aim-100-2_0-yes1-3	0.0	0.0	0.0	0.1	0.0
aim-100-2_0-yes1-4	0.0	0.0	0.0	0.0	0.0
aim-100-3_4-yes1-1	0.1	0.1	0.0	0.1	0.1
aim-100-3_4-yes1-2	0.0	0.2	0.0	0.3	0.4
aim-100-3_4-yes1-3	0.1	0.1	0.3	0.4	0.3
aim-100-3_4-yes1-4	0.3	0.4	0.5	0.5	0.8
aim-100-6_0-yes1-1	0.8	0.9	1.0	0.8	1.0

Table 6: A sample table, taken from the basic GLSSAT results

10 Appendix C: Papers published or submitted

1. Mills, P. & Tsang, E.P.K., Guided Local Search applied to the satisfiability (SAT) problem, Proceedings, 15th National Conference of the Australian Society for Operations Research (ASOR'99), Queensland, Australia, pages 872-883, July 1999.

2. Mills, P. & Tsang, E.P.K., Guided Local Search for solving SAT and weighted MAX-SAT problems, Journal of Automated Reasoning, Special Issue on Satisfiability Problems, Kluwer, 205-223, Vol.24, 2000.

3. Mills, P., Tsang, E.P.K. & Ford, J., Applying an extended Guided Local Search to the Quadratic Assignment Problem. Submitted to Special Edition of Annals of Operations Research (Postworkshop Proceedings of CPAIOR'01), 2001.