Investigating Multiple Noise Parameters in
Stochastic Local Search

Peter W. Nightingale
3rd Year Project Report

October 2001 - March 2002
Abstract

Walksat has proven to be an excellent method for solving large, difficult instances of the propositional satisfiability problem (SAT). It requires a user-specified noise parameter, which if set suboptimally can severely harm performance. It is possible that sets of clauses within a SAT instance have significantly different noise requirements, and as a result, the instance would perform badly with one noise parameter. This report addresses multiple noise parameters (each assigned to a set of clauses) as an idea for speeding up local search.

The use of multiple noise parameters requires some division of the clauses into sets. A likely candidate is the sets generated by encoding other problems into SAT. Two encoded problems are investigated and compared.
Contents

1 Introduction ......................................................... 5
  1.1 Efficiency of SAT solvers ........................................ 5
  1.2 Systematic vs. Local Search ..................................... 6
  1.3 Stochastic Local Search ......................................... 6
    1.4 Walksat .......................................................... 7
      1.4.1 Restarts ..................................................... 7
      1.4.2 Depth, mobility and coverage .............................. 8
      1.4.3 The objective function .................................... 8
      1.4.4 best .......................................................... 8
      1.4.5 tabu ......................................................... 9
      1.4.6 novelty ...................................................... 9
      1.4.7 rnovelty ..................................................... 9
      1.4.8 Comparison of the heuristics ............................. 10

2 Experimenting with SLS ............................................ 11
  2.1 Measuring search cost .......................................... 11
  2.2 Choice of problems and instances .............................. 11
    2.2.1 Averaging over many problem instances .................... 12
    2.2.2 Variables shared between clause types .................... 12
    2.2.3 Leighton graph colouring instances ........................ 13
    2.2.4 Round robin instances ..................................... 13
  2.3 Problem encoding ............................................... 14
    2.3.1 NB-SAT to SAT .............................................. 14
    2.3.2 Graph Colouring to NB-SAT ................................ 16
    2.3.3 Round Robin Tournament Scheduling to SAT ............... 16
    2.3.4 Sparsity ..................................................... 16
  2.4 Purpose of this project ......................................... 19

3 Tools ................................................................. 21
  3.1 Walksat .......................................................... 21
  3.2 Post-processing ................................................ 22
  3.3 Problem encoding ............................................... 23
    3.3.1 Col-NB ...................................................... 23
    3.3.2 NB-B2 ....................................................... 24
  3.4 Problem instance generation with rrgen ........................ 24
4 Kernel noise irrelevance with best and tabu .......................... 26
4.1 The best heuristic .................................................. 26
4.1.1 Kernel noise irrelevance and a theorem to account for it .......... 26
4.1.2 How widely does this theorem apply? .......................... 28
4.2 The tabu heuristic .................................................. 28
4.2.1 Is kernel noise irrelevant? ....................................... 29
4.2.2 The behaviour at low kernel noise .............................. 29
4.2.3 Difficulty and the noise parameters ............................. 32
4.2.4 Conclusions ..................................................... 32

5 An investigation of (r)novelty ......................................... 34
5.1 The novelty heuristic ................................................ 34
5.1.1 Graph colouring ................................................ 34
5.1.2 Better performance where kernel noise is 1 ..................... 37
5.1.3 An alternative encoding for graph colouring ................. 38
5.1.4 The round robin problem ....................................... 39
5.1.5 Comparison of round robin to graph colouring .............. 42
5.1.6 Round robin with high kernel noise .......................... 42
5.1.7 Round robin conclusion ....................................... 45
5.2 The rnovelty heuristic .............................................. 45
5.2.1 Graph colouring .............................................. 45
5.2.2 The round robin problem ..................................... 47

6 Future Directions .................................................... 50
6.1 Tuning best to graph colouring .................................... 50
6.1.1 Faster processing of kernel clauses ............................ 50
6.1.2 New initializations ............................................ 50
6.1.3 Conclusions about new initializations ......................... 53
6.2 Tuning tabu to graph colouring ................................... 54
6.3 A modified rnovelty .............................................. 55
6.4 Other ways of dividing the clauses ............................... 57
6.4.1 Examine the clause length .................................... 57
6.4.2 Look at the distribution of variables over clauses .......... 58
6.4.3 Look at the characteristics of the clauses in the context of structure 58

7 Discussion and Conclusions .......................................... 59
7.1 Multiple noise parameters with encoding-based clause types .... 59
7.2 Possible future directions ......................................... 59
List of Figures

2.1 Illustrating independent sub-instances ........................................ 13
2.2 A small graph example ............................................................. 16
2.3 A round robin solution where \( n = 4 \) ........................................ 17
2.4 Unary and binary encoding move sequences ................................ 19

4.1 Median flips, instance le450_25a, coloured with 25 colours, \textit{tabu} .................. 30
4.2 Median flips, instance le450_15a, coloured with 16 colours, \textit{tabu} .................. 30
4.3 Median flips, instance le450_5d, coloured with 5 colours, \textit{tabu} .................. 31

5.1 Median flips, graph le450_25a, coloured with 25 colours .................... 35
5.2 Equal noise cross-section of Figure 5.1 ........................................ 35
5.3 Median flips, graph le450_15a, coloured with 16 colours .................... 36
5.4 Median flips, graph le450_5d, coloured with 5 colours ....................... 37
5.5 Round robin instance where \( n = 10 \), single noise parameter ................. 39
5.6 Varying a noise parameter while others are constant, \textit{novelty} ............... 40
5.7 A second sample of some charts in Figure 5.6 .................................. 41
5.8 Charts for graph colouring instance le450_15a .................................. 42
5.9 Varying kernel noise parameters together, \textit{novelty} ............................ 43
5.10 Round robin instance where \( n = 12 \), single noise parameter ................. 44
5.11 Median flips, graph le450_25a, coloured with 25 colours, \textit{rnovelty} ............ 46
5.12 Median flips, graph le450_15a, coloured with 16 colours, \textit{rnovelty} ............ 46
5.13 Median flips, graph le450_5d, coloured with 5 colours, \textit{rnovelty} ............. 47
5.14 Round robin where \( n = 10 \), single noise parameter .......................... 48
5.15 Varying a noise parameter while others are constant, \textit{rnovelty} .............. 49

6.1 Comparison of unsatisfied clauses during run ................................... 52
6.2 Probability of the second-best variable being chosen in two cases .......... 56
6.3 Instance le450_25a, solved with modified \textit{rnovelty} .......................... 56
List of Tables

1.1 Solution of instance uf100-01.cnf from collection uf100-450 taken from SATLIB .......................................................... 6
2.1 Three graph colouring instances .......................................................... 14
3.1 Some command line arguments of Walksat ........................................... 21
3.2 The command line arguments of NB-B2 .............................................. 24
3.3 The command line arguments of rrgen ............................................... 25
4.1 Actions of tabu on kernel clauses where both variables break one clause .... 31
5.1 Results for novelty (and comparison with best) .................................... 35
5.2 Results for novelty and rnovelty ..................................................... 48
6.1 Results for colouring instances, with different initializations ............... 51
6.2 Number of colours assigned to each node after 5179 flips .................. 53
Chapter 1

Introduction

The propositional satisfiability problem (commonly called SAT) belongs to the NP-complete class of problems. It is the task of finding an assignment over the variables of a propositional expression, such that the expression takes the value True. By way of example:

You are organizing a house party. You want to have a barbecue, music and punch (b, m, and p respectively). Anne is coming to the party but doesn’t like the smell of barbecues. She asks that there be no barbecue, or punch as a compensation. Barry is also invited. He hates loud music and punch, and asks that you omit one of the two. Charlie loves punch and barbecues and asks for at least one of those.

This instance of SAT can be formulated as determining a satisfying assignment to the following propositional expression:

\[(\neg b \lor p) \land (\neg m \lor \neg p) \land (p \lor b)\]  \hspace{1cm} (1.1)

In this case, a satisfying assignment would be b → True, m → False, p → True, or in English, there is a barbecue and punch but no music, which clearly satisfies the three fussy guests. In the propositional expression 1.1, there are three clauses, representing (from left to right) the constraints placed by Anne, Barry and Charlie. The three clauses form a conjunction. Each clause contains a disjunction of literals (such as \(\neg b\)) over a finite set of variables (in this case \(\{b, m, p\}\)), in their normal or negated forms. This form is known as conjunctive normal form, or CNF. Ordinarily, the variables are Boolean, but this is not necessarily the case [1]. CNF often allows a satisfactory encoding of a problem.

1.1 Efficiency of SAT solvers

For a SAT instance with n variables and of length x, testing a proposed assignment takes time \(O(x)\), and enumerating all possible assignments takes time proportional to \(2^n\). Hence the brute force approach of testing every possible assignment takes exponential time. Consider a SAT instance with 100 variables. There are \(2^{100}\) assignments to the variables, approximately \(10^{30}\). Assuming solutions are few and widely spread, a large fraction of the possible assignments would have to be tested. To test all \(2^{100}\) assignments at a rate of \(10^{12}\) assignments per second would take more than \(10^{16}\) years. However, state-of-the-art solvers can do considerably better than that, as shown in Table 1.1. The example used is one generated randomly, with parameters chosen such that the example is hard to solve for one of its size.
Table 1.1: Solution of instance uf100-01.cnf from collection uf100-450 taken from SATLIB [3].

<table>
<thead>
<tr>
<th>Solver</th>
<th>Type of solver</th>
<th>Time taken to solve (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Walksat with best heuristic, noise=50%</td>
<td>Local Search</td>
<td>0.008</td>
</tr>
<tr>
<td>SATO</td>
<td>Systematic</td>
<td>0.01</td>
</tr>
</tbody>
</table>

1.2 Systematic vs. Local Search

Traditionally systematic search procedures have been used to solve SAT. The most effective of these are recent improvements on the Davis-Putnam-Logemann-Loveland procedure [4]. These solvers systematically traverse a tree, of which the internal nodes represent partial assignments and the leaves represent complete assignments over the variables of the problem instance. An algorithm following a systematic approach is complete in the sense that all problem instances can be solved or proven to be unsatisfiable, given sufficient time. The SATO solver (Table 1.1) is of this type, and performs well. However, if an instance of SAT is likely to be solvable, relaxing the completeness property can give better performance. This observation caused the development of many local search procedures. They initialize the search at some place in the space of total assignments (often randomly chosen), and simply wander around guided by local knowledge. Only tabu (section 1.4.5) keeps a record of where the search has covered, and then only a limited record in the form of a chronological list of the moves made. If a full record were kept, such a data structure would become huge and impractical to search, and so no test is performed for cycles in the search or poor coverage of the search space. Many heuristics have been developed which exploit various indicators in the local state, in various ways. Since local search can quickly descend into local minima (near-solution points) in the search space, it is particularly well-suited to the optimization variant of SAT named MAX-SAT, where the objective is to maximize the number of satisfied clauses (possibly within a time constraint).

1.3 Stochastic Local Search

Stochastic Local Search (SLS) is an umbrella term for local search procedures that employ randomness in some way. The introduction of randomness into the process of local search provides a mechanism for escaping from the local minima of the search space, by making moves that appear sub-optimal when considered with only the local knowledge. The level of randomness (or noise) exhibited by the search is generally controlled by a parameter of the whole search, for example in simulated annealing this is the temperature, in GSAT [5] it is the random walk parameter, in Walksat it is the noise parameter (which is used in different ways according to the heuristic employed; see section 1.4) and in tabu search it is the length of the tabu list (the only form of tabu search considered in this project is Walksat with the tabu heuristic). Of these, only Walksat picks a clause, then a variable to flip. The others pick variables directly, so multiple noise parameters for different clause types could not be applied to them.

It is well known that the setting of the noise parameter is critical in the performance of SLS procedures [6], with a suboptimal noise setting causing search time several orders of magnitude longer than optimal. Across suites of similar SAT problem instances, the optimal noise level is often approximately uniform but it is certainly not uniform across all SAT problem instances. Frisch and Peugniez [1] show the variation of optimal noise level when
encoding NB-SAT problem instances (that is SAT with non-Boolean variables) with various
domain sizes. In their study, using Walksat with the best heuristic and negative unary/unary
(UU, later referred to simply as unary) encoding of NB-SAT to SAT, they find instances with
a domain size of 2 have an optimal $p$ (noise) setting of 0.550, and those with a domain size of
32 have an optimal $p$ (noise) setting of only 0.010. $p$ is a probability, hence this variation is
of more than half of its range.

McAllester, Selman and Kautz [6] claim that the optimal noise parameter can be inferred
from statistics collected either with several brief runs, or collected during the run (and being
used to dynamically adjust the noise parameter). However, the validity of the “invariants”
discovered by McAllester et al. is refuted by Anthony Doggett in his project [7], citing various
reasons, among them that sampling the Optimality Invariant (one of the two invariants claimed
by McAllester et al.) was problematic, and that the invariants don’t hold for all SAT instances.

The purpose of this project is to investigate the possibility of categorizing clauses according
to the type of constraint they represent, and assigning the noise parameter not to the whole
instance, but to categories independently, in effect localizing the noise parameter. An example
of clause categorization is graph colouring instances encoded as SAT; if the negative unary
encoding 2.3 were used, then there would be two categories: kernel and at-least-one types.
Results presented later in this paper show that in one experiment (using the best heuristic)
the noise assigned to the kernel category has no effect whatsoever on the solution time!

1.4 Walksat

The focus of this project is Walksat (also shown in table 1.1) since it shows good performance
and is adaptable to this new strategy of using multiple noise parameters. Algorithm 1 illus-
trates the Walksat structure (first proposed by Selman, Kautz and Cohen [5]). Walksat is
a form of Stochastic Local Search, which means it keeps no record of where it has searched
(with the exception of the taboo heuristic which keeps a very limited record) and considers only
the current position when deciding the next move. This is also known as Total Assignment
Search, because at any point the algorithm has a total assignment over the variables. Walksat
starts with a random assignment over the variables, and its progress from there is determined
by the heuristic used and the random choices of clauses.

1.4.1 Restarts

This is the technique of starting the search again from a random assignment, after some
number of flips $t'$. Restarts do not affect the behaviour of Walksat when the noise setting(s)
are optimal, since (intuitively) both noise and restarts are mechanisms for stopping the search
becoming stuck in local minima. If the noise mechanism is working correctly, there is no need
for restarts.

‘for approximately optimal and larger-than-optimal noise settings, these SLS algo-
rithms [including Walksat with best, tabu, and novelty] are essentially memoryless,
as for a given time $t$, restarting at time $t' < t$ does not significantly influence
the probability of finding a solution in time $t$. Consequently, for these algorithms
random restart is ineffective.’ Hoos and Stützle [12]

Hoos didn’t consider novelty but it is very similar to novelty, so I suspect that their conclusion
extends to novelty. This project is concerned with performance at the optimal noise setting,
and the location of the optimal settings of multiple parameters, and restarts would make no
difference to either of these. All experiments for this project were performed without restarts.

1.4.2 Depth, mobility and coverage

The search characteristics depth, mobility and coverage were proposed by Schuurmans and
Southey [8]. Definitions of search characteristics are of benefit when attempting to analyze
the effects of noise variation on search times. The three characteristics ‘are conflicting de-
mands, and successful methods [of local search] are primarily characterized by their ability to
effectively manage the tradeoff between these factors’ [8, section 1]. They are defined below:

- Depth measures the average number of unsatisfied clauses during the search. To avoid
  biasing the number due to the initial random assignment, the first $t$ steps are not included
  in the average (where Schuurmans and Southey used $t = 100$). Since a solution contains
  no unsatisfied clauses, minimizing depth is desirable.

- Mobility measures how rapidly the search traverses the search space. This is calculated
  from the Hamming distance\(^1\) between variable assignments that are $k$ search steps apart.
  This quantity is averaged over the entire sequence. Moving quickly through the assign-
  ment space is clearly better than moving slowly, so maximizing mobility is desirable.

- Coverage measures how systematically the search explores the assignment space. First
  the size of the largest unexplored gap is estimated, where a gap is the Hamming distance
  between an unexplored assignment and the nearest explored assignment. The rate of
  reduction of the largest gap size is the coverage rate. Note that it is good to maximize
  this rate since this indicates that the search is covering new areas of the assignment
  space as it runs.

1.4.3 The objective function

As shown in step three of algorithm 1, the variable to be flipped (changed from true to false
or vice versa) is chosen by a heuristic, four of which are presented below. In the descrip-
tions below, the term breaks refers to the number of satisfied clauses that would become unsat-
sified by the flipping of a variable. makes – breaks is the number of unsatisfied clauses that would
become satisfied minus the number of satisfied clauses that would become unsatisfied (so this
function has a stronger pull towards areas with a high proportion of satisfied clauses than
breaks alone). These are known as objective functions because they attempt to move the
search closer to its objective (all clauses satisfied).

1.4.4 best

The best heuristic is the original strategy suggested by Selman, Kautz and Cohen [5], and
hence is sometimes known as SKC and also as WSAT. If any variables in the chosen clause
break no clauses when flipped, the procedure chooses a variable randomly among those variables. Failing
that, with probability $p$ the procedure chooses a variable at random from those contained in
the clause, otherwise a variable is chosen that minimizes breaks.

\(^1\)The Hamming distance between two variable assignments in the search space is simply the number of
variables that are assigned different values.
Algorithm 1 Walksat

1. Pick an assignment at random
2. Pick an unsatisfied clause at random
3. Flip a variable contained in that clause according to the heuristic
4. Test formula for satisfaction
5. If unsatisfied, return to 2 (until cutoff reached). If satisfied, succeed.

1.4.5 tabu

Similarly to best, tabu limits its choice to variables that break no clauses. If no such variables exist, the choice is made as follows: a list is kept of the variables flipped in the previous $t$ steps; tabu refuses to flip a variable that is in the list. From the remaining variables, one is chosen that minimizes breaks. If all the variables in the chosen clause are tabu (i.e. were flipped within $t$ steps) then another unsatisfied clause is chosen. If all variables in all unsatisfied clauses are tabu, the list is ignored. The noise parameter here is the size of $t$: a size of one will break deterministic loops that flip the same variable repeatedly, a size of two breaks a loop that flips two variables alternately, etc.

1.4.6 novelty

This heuristic uses a different objective function. Variables are sorted by makes − breaks. It considers the best and second-best variables; if the best is not the most recently flipped variable, it is chosen. Otherwise, the second-best variable is chosen with probability $p$ and the best with probability $(1−p)$. This can be thought of as adding a tabu list of length one to a probabilistic choice. With probability $p$, the tabu list is used (intuitively to stop the repeated flipping of one variable) and with probability $(1−p)$ it is ignored.

1.4.7 mNovelty

This is the same as novelty, except in the case where the best variable is also the most recently flipped. In that case, where $p$ is the noise parameter and $n$ is the difference between the scores of the best and second-best variables:

```plaintext
WHERE p<0.5
    WHERE n=1
        WITH probability 2p pick the second best, else pick the best
    WHERE n>1
        pick the best variable
ELSE
    WHERE n=1
        pick the second-best variable
    WHERE n>1
        WITH probability 2(p-0.5) pick the second-best, else pick the best
```
Since this heuristic only considers the best and second-best variables under the sort, it is sometimes liable to get stuck in local minima. To break deterministic loops in the search, the variable is chosen randomly every 100 flips. The intuition is that a large difference in the goodness of the best and second-best variables should favour the best. If the difference of the \textit{makes} – \textit{breaks} function is only one however, the second-best is favoured to aid mobility and coverage.

1.4.8 Comparison of the heuristics

It is interesting that \textit{best} and \textit{tabu} sort variables by \textit{breaks}, and (r)\textit{novelty} (meaning both \textit{novelty} and \textit{rnovelty}) sort variables by \textit{makes}–\textit{breaks}. If \textit{best} is written to use \textit{makes}–\textit{breaks}, it performs poorly. It seems that \textit{makes} – \textit{breaks} is too “strong” for \textit{best}, in that \textit{best} is not capable of achieving mobility and coverage with the \textit{makes} – \textit{breaks} function. Also, if \textit{breaks} is used with (r)\textit{novelty}, the search has inadequate depth (defined in section 1.4). \textit{makes} – \textit{breaks} is better at targeting the search towards areas where most of the clauses are satisfied. (r)\textit{Novelty} have more sophisticated handling of noise than \textit{best}, allowing them to take full advantage of the stronger objective function.

The heuristics may also be grouped by how they handle noise. \textit{Best}, \textit{novelty} and \textit{rnovelty} all use a probability, representing (in various ways) the chance of making a move against the flow towards minima. Hence, a high probability equates to more noise. The \textit{tabu} heuristic has no such probability, but a list length that determines how many flips must separate two flips of the same variable. Clearly a longer tabu list causes more moves against the objective function during the search.

This project covers all four of the heuristics described above.
Chapter 2

Experimenting with SLS

2.1 Measuring search cost

The difficulty involved in finding a solution to a problem instance is known as the *search cost*. It is commonly measured as the time taken, or the number of moves made by the algorithm (for Walksat, the number of flips). For Walksat, either of these quantities need to be averaged over a number of runs, because of its randomized nature. If the sample is too small, the results will appear to be erratic. In this project, I have used the number of flips as the measure, because measuring the time accurately is problematic. It requires that the Walksat program have a constant share of processor time and memory across all the runs performed on every instance. The best way to do this would be to ensure that no other programs are running on the machine used for the runs. Also, identical machines should be used for every run. The number of flips, on the other hand, is independent of machine architecture and speed (making experimentation much less difficult). However, it does mask the differences in flip rate between instances and heuristics.

Walksat is prone to outliers; the algorithm is capable of solving an instance very quickly by only flipping the variables required to get to a solution. It is also capable of running forever, covering the same assignments repeatedly. The mean average is affected by outliers, and for it to be calculated the cutoff of the search needs to be set so high that *all* runs succeed. This could be very expensive, particularly since many experiments will involve suboptimal noise parameters, and could take a very long time.

The solution to this problem is to use the median average. It is not affected greatly by outliers, and its calculation only requires the shorter half of the runs to be successful, which should save a large amount of CPU time.

2.2 Choice of problems and instances

Initially, the intuition for multiple noise parameters comes from work by Frisch and Peugniez [1]. They considered graph colouring instances with various domain sizes, and found when encoding as SAT (using the unary encoding) then solving, the optimum noise level decreased as domain size increased. This led to the conjecture that different types of clauses generated by the encoding could have different noise requirements, and as their size and numbers change because of the increasing domain size, the overall noise requirement decreases. If the two types have different noise requirements, could two noise parameters provide better performance?
To test the hypothesis that multiple noise parameters may speed up local search, SAT instances whose clauses can be divided into types are required. The most obvious answer is to look at encoded problems, such as graph colouring which can be encoded into SAT with two clause types of significantly different forms. Although there are other ideas that could be considered for dividing the clauses (section 6.4), only encoding-related types will be considered in this project, because it seems the most promising (the division of the clauses is natural; there is no need to find some way of dividing them) and certainly involves enough work for one project. The word graph will be used exclusively to describe the graphs of graph colouring. Visual data displays will be called charts in this report.

The graph colouring problem is commonly encoded into SAT as two clause types, which makes it convenient for experimentation. With only two noise parameters (and a feasible instance of graph colouring), the parameter space can be searched exhaustively. The round robin problem was chosen for the opposite reason: the encoding has five clause types, which could increase the possibility of finding some interesting characteristics, or indeed a clause type that requires a significantly different noise setting to the others for the search to perform well. Some of the round robin clause types are similar to graph colouring ones, but there are also longer kernel clauses (for a more detailed treatment of this, see section 2.3.3). It is known that as instances of SAT get harder, the setting of the noise parameter becomes more critical [7]; the chart of median flips against the noise parameter becomes steeper. It will be interesting to see which clause types dictate the steepness of the chart, or if they all contribute. Charts of search cost against the noise parameter of a specific clause type (where the other noise parameters are constant) will reveal how critical the setting of that noise parameter is.

2.2.1 Averaging over many problem instances

Averaging over a suite of problem instances can give a general picture of performance at various noise parameters. The round robin problem has a single instance for each value of its parameter (the number of teams), so averaging is impossible. Graph colouring could be averaged, but this has the potential to hide the details of each instance among the others; and also the graph colouring instances must be generated to be suitably similar (because the average of dissimilar results can be misleading). For these reasons, I have used common test instances of different sizes and difficulties, without averaging.

2.2.2 Variables shared between clause types

Typically, problem instances encoded as SAT tend to have all their variables shared between clause types, because those types represent different constraints over the set of variables. In graph colouring encoded as described in sections 2.3.2 and 2.3.1, all the variables contained in the ALO clauses are also present in the kernel clauses. However, if two clause types are allowed to have no shared variables, an example can be easily contrived that proves multiple noise parameters can improve the speed of Walksat! This is done by simply taking two SAT instances with considerably different optimal noise parameters, and joining them into one instance by conjoining their clauses (without sharing variables). These will be referred to as sub-instances to differentiate them from sets of clauses that share variables. Figure 2.1 illustrates the difference. Any noise setting covering both sub-instances will then be suboptimal for at least one, but two noise parameters will allow the optimal settings.

The behaviour of the solver on one sub-instance is completely independent of its behaviour
on the other except with the tabu heuristic. Tabu is different because the tabu list is shared between the two sub-instances, so each sub-instance essentially sees a tabu list of varying length, shorter when most moves are being made in the other sub-instance. The other heuristics maintain independence because flipping a variable in one sub-instance can neither make nor break any clauses in the other sub-instance, and it is only makes and breaks (as the objective function) that influence the behaviour of Walksat.

So there is a theoretical case where multiple noise parameters would be essential in solving a SAT instance quickly. But does this generalize to real-world problems? It is easy to imagine planning problems with only one variable shared between sets of clauses. Consider building a bridge: the structure can’t be built until the foundations have been dug, and it makes sense to dig the foundations on each side in parallel. The only variable shared between the planning of the digging and the planning of the building, is the time that the digging will end. Once that variable is set (and not changed again during the rest of the search) the two parts become essentially independent. If this problem were encoded to SAT, the shared variable may be represented by a set of Boolean variables, that are shared between clause sets (depending on the encoding).

This type of problem will not be considered in this project, but it does illustrate the potential of multiple noise parameters. This could be extended to a small set of shared variables, the middle case in Figure 2.1. This, and other ways of dividing the clauses into coherent types, are discussed further in section 6.4.

2.2.3 Leighton graph colouring instances

The graph colouring problem instances used in this project were picked from the Dimacs collection [9]. They were picked to have a range of chromatic numbers, and difficulty level that allows good results with a cutoff of 100 000 flips. The median was typically taken of 101 runs. These graphs are well accepted as test instances in research of this kind. They are listed in Table 2.1.

2.2.4 Round robin instances

The round robin tournament scheduling problem is that of arranging teams into games, for a tournament in which each team plays each other. For small numbers of teams, it is easy. It does get rapidly harder, though, because ‘the combinatorics are explosive’ [10]. At 14 teams,
<table>
<thead>
<tr>
<th>Problem</th>
<th>Total colours used</th>
<th>Chromatic number</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>le450_25a</td>
<td>25</td>
<td>25</td>
<td>450</td>
</tr>
<tr>
<td>le450_15a</td>
<td>16</td>
<td>15</td>
<td>450</td>
</tr>
<tr>
<td>le450_5d</td>
<td>5</td>
<td>5</td>
<td>450</td>
</tr>
</tbody>
</table>

Table 2.1: Three graph colouring instances

the problem is too hard for my purposes (having a median of approximately 400 000 flips) and some experiments showed that 12 teams is also too hard. I chose to use 10 teams, with a cutoff of 100 000 flips, and taking the median of 201 attempts.

2.3 Problem encoding

Since SAT is NP-complete, all other NP-complete problems may be encoded as SAT in polynomial time and then solved. Efficient SAT solver implementations therefore benefit many areas of computer science. However, there are many possible encodings of each problem into SAT; Hoos [2] investigates possible encodings for constraint satisfaction instances, and comes to the general conclusions that ‘it seems to be much more advisable to use sparse rather than compact encodings’ (‘compact’ meaning comprising fewer variables), and that solving constraint satisfaction instances directly yields a ‘surprisingly small advantage... which is outweighed by other advantages of the generic SAT-encoding and solving approach such as the availability of very efficient implementations’. For the bulk of this project, I consider only the most efficient known encoding of problems, since the hope of the project is an improvement upon the state-of-the-art speed.

Polarity is important for the arguments in Chapter 4. If a set of clauses all contain only negative literals, the whole set can be thought of as negative, and the same for positive. Thus a set of clauses has a polarity. In a unary encoding of many problems (including graph colouring and round robin), the kernel clauses (even if there are many types) have a uniform polarity, and are accompanied by ALO or AMO clause types with the opposite polarity.

2.3.1 NB-SAT to SAT

One encoding that is important for this project is from NB-SAT (or SAT with non-Boolean variables) to SAT. A non-Boolean variable takes a value from a domain that has size not equal to two. For example, if non-Boolean variable $A$ has the domain $\{a, b, c\}$ then expression 2.1 is a valid non-Boolean literal, and takes the value True if and only if the variable $A$ is equal to $a$ or $b$.

\[ A \in \{a, b\} \]

\[ (A \in \{a, b\} \lor \neg B \in \{c\}) \land (A \in \{a, b\} \lor B \in \{b\}) \]

2.2 is an NB-CNF expression, which is very similar in form to a Boolean CNF expression, with only the form of the literals differing. There are two well-known ways of encoding the formula, unary and binary. With the unary encoding, each CNF variable maps to one NB-CNF variable-value pair. For example, the literal $A \in \{a, b\}$ translates to $A/a \lor A/b$, where
$A/a$ represents the Boolean literal that maps to the NB-CNF variable $A$ having value $a$. To encode the constraints represented by 2.2 to CNF, three classes of conjunct are required (according to Frisch and Peugniez [1]):

1. Kernel - these map from the clauses of the NB-CNF expression, with a bijective relationship. Assuming the domains of $A$ and $B$ are both $\{a, b, c\}$, expression 2.2 maps onto expression 2.3. The term negative in this context means translating $\neg B \in \{c\}$ into $\neg B/c$ rather than changing it into the positive (i.e. non-negated) form, $B/a \lor B/b$. The negative form is shorter and is a richer encoding.

\[
(A/a \lor A/b \lor \neg B/c) \land (A/a \lor A/b \lor B/b) \tag{2.3}
\]

2. ALO (at least one) - these represent the constraint that each NB-CNF variable must have at least one value assigned to it. Expression 2.2 requires the ALO clauses shown in 2.4. There is one clause per variable, and one literal per clause for each value in the domain of the variable.

\[
(A/a \lor A/b \lor \lor A/c) \land (B/a \lor B/b \lor B/c) \tag{2.4}
\]

3. AMO (at most one) - each NB-CNF variable must not be assigned more than one value. The AMO clauses for 2.2 are shown in 2.5. These function by eliminating the possibility of pairs of values, for all possible pairs.

\[
(\neg A/a \lor \neg A/b) \land (\neg A/a \lor \neg A/c) \land (\neg A/b \lor \neg A/c) \land \\
(\neg B/a \lor \neg B/b) \land (\neg B/a \lor \neg B/c) \land (\neg B/b \lor \neg B/c) \tag{2.5}
\]

The encoded formula is simply the conjunction of 2.3, 2.4 and 2.5. With the unary encoding, it is possible for the solver to make assignments that do not correspond to an assignment of the non-Boolean variables in the original formula. The non-Boolean variable $A$ with domain $\{a, b, c\}$ corresponds to the Boolean variables $\{A/a, A/b, A/c\}$. Some assignments over the Boolean variables are shown below.

<table>
<thead>
<tr>
<th>$A/a$</th>
<th>$A/b$</th>
<th>$A/c$</th>
<th>Meaning in terms of $A$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>True</strong></td>
<td><strong>False</strong></td>
<td><strong>False</strong></td>
<td>$A \rightarrow a$</td>
</tr>
<tr>
<td><strong>False</strong></td>
<td><strong>False</strong></td>
<td><strong>False</strong></td>
<td>$A \rightarrow \text{null}$</td>
</tr>
<tr>
<td><strong>True</strong></td>
<td><strong>True</strong></td>
<td><strong>False</strong></td>
<td>$A \rightarrow a \lor b$</td>
</tr>
</tbody>
</table>

The fact that assignments with inconsistent meanings are present appears to speed up local search. Of the examples above, only the first row has a consistent meaning, however the last row would be meaningful where the original NB-SAT instance were encoded from graph colouring (see section 2.3.2).

The binary encoding is an alternative to the above. In the binary encoding, each non-Boolean variable is allocated $\lceil \log_2 n \rceil$ Boolean variables, where $n$ is the size of its domain. Each value in the domain is given a unique assignment over the Boolean variables, in a base 2 encoding. If the size of the domain is a power of 2, all assignments over the Boolean variables map to an assignment of the non-Boolean variable, which means ALO and AMO clauses are unnecessary. There are methods of encoding non-Boolean variables with domain sizes that are not powers of two, which are covered by Frisch and Peugniez [1]. They also present results showing that use of the binary encoding leads to longer solution times. Section 2.3.4 gives a full justification of using only the unary encoding.
2.3.2 Graph Colouring to NB-SAT

The graph colouring problem is at the core of many other problems, such as timetabling. A graph is a number of items named nodes with interconnections called edges. The graph colouring problem is that of assigning colours (of which there are a finite number) to the nodes of a graph, such that each edge of the graph connects two different colours. A small graph is shown in Figure 2.2, along with its formal representation.

The encoding of this problem instance to NB-SAT requires the number of colours to be specified. The smallest number of colours $G$ can be coloured in is 3 (this number is known as the chromatic number). In this case, the encoding will be of the 3-colouring decision problem (the problem of deciding whether $G$ can be coloured in 3 colours). The number of colours used in the encoding is referred to as the total colours in this report. Each edge is encoded as 3 clauses in NB-CNF. The colours are $x$, $y$ and $z$. The resulting instance of NB-SAT has 1 variable for each node in the graph. The first edge, $(a, b)$ is encoded as shown in expression 2.6. The $\neg$ symbol is used here to indicate non-membership, so $\neg a \in \{x\}$ is True if and only if $a$ does not equal $x$. All other edges in the graph are encoded equivalently. Literally, each clause of the encoding asserts that two nodes may not both be a certain colour. This encoding is described by Frisch and Peugniez [1], and is the only one I have come across.

\[
(-a \in \{x\} \lor \neg b \in \{x\}) \land (-a \in \{y\} \lor \neg b \in \{y\}) \land (-a \in \{z\} \lor \neg b \in \{z\})
\]  

(2.6)

When encoding the resulting NB-SAT instance into SAT, note that AMO clauses are not required. This is because the AMO clauses constrain the number of assignments of each non-Boolean variable to be 1 or 0. Multiple assignments to a non-Boolean variable are acceptable in graph colouring because they are equivalent to multiple colourings of a node. If a graph is coloured so that all edge constraints are satisfied, and one node has two colours, this indicates that two solutions to the problem instance have been found since either colour could be assigned to the node, and all constraints would remain satisfied.

2.3.3 Round Robin Tournament Scheduling to SAT

The Round Robin problem is that of assigning teams to games, in a tournament where each team plays each other exactly once. Where $n$ is the number of teams, the tournament is played over $n - 1$ weeks, and with $\frac{n}{2}$ pitches. This gives $\frac{n}{2}(n - 1)$ slots, each of which must be assigned two teams. Other constraints are that a team may not play on the same pitch more than twice, and a team plays one game each week of the season (otherwise the games would clash). The problem instance where $n = 4$ is illustrated in Figure 2.3.

Béjar and Manyà [10] proposed an encoding of the above constraints into SAT, with five clause types. The pitch is given a number $i$ where $1 \leq i \leq \frac{n}{2}$. Each pitch has end $r$ where
1 \leq r \leq 2$, and there must be a team assigned to each end for a game to take place. The
week is given a number $j$ where $1 \leq j \leq n - 1$. A specific end of a specific pitch on a specific
week will be called a place. A team must be assigned to each place, and the unary encoding
is used. For each place, each team has a Boolean variable, and in a solution only one of those
variables is assigned to True, indicating the team assigned to the place. $k$ is used to represent
the team, and there is a subtlety in the encoding allowing the omission of $k = n$ when $r = 1$
and of $k = 1$ when $r = 2$. This is explained in (1) below. Each Boolean variable is represented
as $p_{ij}^{k}$ and the set of Boolean variables is given here:

$$\left\{ p_{ij}^{1k} \mid 1 \leq i \leq \frac{n}{2}, 1 \leq j \leq n - 1, 1 \leq k \leq n - 1 \right\} \cup \left\{ p_{ij}^{2k} \mid 1 \leq i \leq \frac{n}{2}, 1 \leq j \leq n - 1, 2 \leq k \leq n \right\}$$

And the set of clauses is given here:

1. **In each slot, one team plays against another team.** This is analogous to the ALO con-
   straint used in the encoding of NB-SAT to SAT (section 2.3.1). If a slot is considered
   as two non-Boolean variables (one for each team playing), then the constraint can be
   encoded as two ALO clauses. In this case however, there is a subtlety (noted above),
   in that one fewer variables is included in each of the two ALO clauses required for each
   slot. If the teams are represented by numbers (1..n), and the ends of the pitch are
   represented as non-Boolean variables $k_{1}$ and $k_{2}$ then the domain of $k_{1}$ is (1..n - 1) and
   the domain of $k_{2}$ is (2..n) since the clauses in (2) add the constraint that $k_{1} < k_{2}$. The
   clauses required are:

   $$(p_{ij}^{11} \lor \ldots \lor p_{ij}^{1n-1}) \land \left(p_{ij}^{22} \lor \ldots \lor p_{ij}^{2n}\right)$$

   These ALO clauses together with the clauses in (3) ensure that each team plays exactly
   one other every week.

2. **In each slot** ($p_{ij}^{1k_{1}}, p_{ij}^{2k_{2}}$) it holds that $k_{1} < k_{2}$. These clauses force the slots into a specific
   form, breaking a symmetry in the problem by ensuring that team $k_{1}$ playing team $k_{2}$ on
   pitch $i$ in week $j$ has only one representation rather than two equivalent ones. It ensures
   that the team with the lower number plays at the end where $r = 1$. This helps simplify
   other clause types. For instance, type (4) would contain four times as many clauses if it
   had to cover the possibility that $k_{1} > k_{2}$. For each two teams $k_{1}$ and $k_{2}$ where $k_{1} > k_{2}$
   we define the clause:

   $$\left(\neg p_{ij}^{1k_{1}} \lor \neg p_{ij}^{2k_{2}}\right)$$

   Figure 2.3: A round robin solution where $n = 4$
3. Every team plays one game in each week of the season. For every week and for every team, it is necessary to ensure that the team plays no more than once (otherwise a timetabling clash has occurred). Combined with filling all n places for every week (ensured by the clauses in (1)), this asserts that each week contains a permutation of teams over places. For each week j and for each team k, for each two fields i_1 and i_2 such that i_1, i_2 ∈ \{1..\frac{n}{2}\} and for r_1 and r_2 such that r_1, r_2 ∈ \{1..2\} the following clauses are defined:

$$-p^{r_1}_{i_1,j} \lor -p^{r_2}_{i_2,j}$$

given that \(p^{r_1}_{i_1,j} \neq p^{r_2}_{i_2,j}\). Also clauses containing variables of the form \(p^{1n}_{ij}\) or \(p^{21}_{ij}\) are not generated since these variables don’t exist in this encoding.

4. Every two teams play each other exactly once. Throughout the tournament, each team must play every other once. It is enough for these clauses to assert the constraint that no two teams should play each other twice, since clause type (1) ensures that every slot is filled. For each two different slots of the form \(\left(p^{1k}_{i_1,j_1}, p^{2k}_{i_2,j_2}\right)\) and \(\left(p^{1k}_{i_1,j_3}, p^{2k}_{i_2,j_4}\right)\), the following clause is defined:

$$-p^{1k}_{i_1,j_1} \lor -p^{2k}_{i_1,j_2} \lor -p^{1k}_{i_2,j_3} \lor -p^{2k}_{i_2,j_4}$$

5. No team plays more than twice in the same field over the course of the season. Each team will play on each pitch, playing one game on one pitch and two games on every other. For each team k, for each field i, for each three different weeks \(j_1, j_2, j_3\) and for each \(r_1, r_2, r_3\) such that \(r_1, r_2, r_3 \in \{1..2\}\), the following clause is defined:

$$-p^{r_1}_{ij_1} \lor -p^{r_2}_{ij_2} \lor -p^{r_3}_{ij_3}$$

Béjar and Manyà ‘have performed experiments with five alternative SAT encodings, but the results obtained were rather worse’ [10]. For this reason, I only consider this encoding here, despite the numerous other encodings that are possible.

This encoding is unary so it has the same issue with multiple and null assignments outlined in section 2.3.1, and similarly to other unary encodings, clause type 1 is known as ALO, and the others are known as kernel clauses.

2.3.4 Sparsity

Hoos [2] found that (for the problems he investigated) a ‘sparse’ (unary) encoding is more computationally competitive than a ‘compact’ (binary) one. The unary encoding has more Boolean variables than strictly necessary, and Hoos claims that this causes Walksat to be better able to move around the search space. Hoce tested his theory with the Constraint Satisfaction Problem (CSP) and Hamiltonian Circuit Problem (HCP), and found that unary came out best in all his tests [2, Table 1 and Figure 1]. I believe that this conclusion applies to the problems considered in this project, because they can both be thought of as CSP with the constraints restricted to particular types.

Intuitively, the unary encoding is much easier to move around when the only move allowed is flipping a single variable, as Figure 2.4 illustrates. The figure shows that (in this particular example) the binary case requires one more move, and passes through states 0 and 4, where
the unary example only passes through a state where there is no assignment. It is assumed that ALO clauses are used in the unary example, and that kernel clauses are negative.

This is significant since states 0 and 4 could be arbitrarily bad in terms of the objective function, so to reach state 5 (let’s assume that state 5 is a required part of the solution), the binary encoding requires that the search move through arbitrarily bad states. The unary encoding only requires that the search move through a state with no assignment, which can only break the one relevant ALO clause. The unary encoding only ever requires two moves, but with the binary encoding the number of moves equals the hamming distance between the encodings of the two states. ‘Apparently, the compact [binary] encoding induces rather flat, featureless search spaces which impede local search and local minima escape’ [2]. Also the flip rate achieved by Walksat is poorer with the binary encoding [2, 1]. For these reasons, I will be using the unary encoding throughout this project.

2.4 Purpose of this project

This project investigates the idea of applying different noise settings to the different types of clause produced by encoding problems to SAT. The main hope is to improve performance. Chapter 4 looks at the best and tabu heuristics, and proves that best is totally unaffected by the setting of kernel noise parameters. This means that only one of the two noise parameters in an encoded graph colouring instance has any effect, and only one of the five types in an encoded instance of round robin. Hence, using multiple noise parameters is only as effective as using one. The chapter also shows that tabu behaves in a similar way to best, and as a result it is unlikely that multiple noise parameters will provide a significant benefit for tabu.

Some ideas for improving the performance of best (given the property mentioned above) are
presented in section 6.1, one of which (special initializations for unary SAT-encoded problems) applies to all Walksat heuristics, and indeed I speculate that it applies to all local search procedures. Also, tabu can be tuned to unary SAT-encoded problems, and some ideas for doing this (applied specifically to graph colouring) are presented in section 6.2.

Chapter 5 investigates the properties of the novelty and rnovelty heuristics with multiple noise parameters. For novelty, two instances of graph colouring show a marked decrease in median flips when one of the noise parameters (the kernel noise) is set to 1, and the third shows no significant difference. For rnovelty, the performance is awful for graph colouring when kernel noise is set below 0.5. This is investigated further in section 6.3, and a modified rnovelty is presented without this characteristic. For the round robin problem, (r)novelty shows no promise of improvement by using multiple noise parameters.

Chapter 6 is about possible ways of extending this research. Three sections of this chapter have been covered above because they were related to specific heuristics. The fourth part (section 6.4) is about using multiple noise parameters, but dividing the clauses in another way not related to an encoding. Three possible methods are presented.
Chapter 3

Tools

3.1 Walksat

The Walksat tool was written by Henry Kautz and Bram Cohen to support their work. The structure it is based on is described by Selman et al. [5], and it is distributed via SATLIB [3]. It was chosen for this project because it is fast, well studied and well developed, and is easy to use, and not prohibitively difficult to modify. It generates a good range of statistics, some of which are described below.

The heuristics described in section 1.4 are implemented in the Walksat tool, along with one other (random) which has no noise parameter and is therefore not considered in this project (it also performs poorly on most SAT instances). Walksat takes a set of command line arguments, and a problem instance file on standard input. Statistics and solutions are printed to standard output. Some of the more useful command line arguments are shown in Table 3.1.

The noise parameter for Walksat is normally specified as a command line argument, to apply to all clauses. For this project, a way is needed of specifying the noise parameters of each type of clause independently. Since the finest granularity that could possibly be required is to specify each clause an individual noise parameter, this is the approach taken. It is also simpler to attach a noise parameter to each clause than attach a token to each clause and a noise parameter to each type of token. The original input file format is shown below:

c  
c Comment lines at the beginning of the file start with ’c’
c

<table>
<thead>
<tr>
<th>Argument</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>-cutoff N</td>
<td>Set the maximum number of flips in one try</td>
</tr>
<tr>
<td>.tries N</td>
<td>Set the maximum number of tries</td>
</tr>
<tr>
<td>-init FILE</td>
<td>Initialise the variables according to FILE</td>
</tr>
<tr>
<td>-best, -tabu, -novelty, -rnovelty</td>
<td>Set the heuristic to be used</td>
</tr>
<tr>
<td>-sample N</td>
<td>Print the value of the objective function every N flips</td>
</tr>
<tr>
<td>-trace N</td>
<td>Print general statistics every N flips</td>
</tr>
</tbody>
</table>

Table 3.1: Some command line arguments of Walksat
p cnf <num_atoms> <num_clauses>
lit1 lit2 ... litN 0
...

There is a line of literals specifying the contents of each clause, preceded by a header that specifies the number of atoms and clauses the solver should expect. Each line of literals is terminated by a zero, so I simply added the noise parameter after that, and another 0 to end the line:

lit1 lit2 ... litN 0 numerator denominator 0

Or, for tabu, where the parameter is not in the form \[
\text{numerator denominator}
\]

lit1 lit2 ... litN 0 list_length 0

This format is backwardly compatible, because the unmodified Walksat only reads the line to the first zero. Other solvers aren’t guaranteed to accept this format however. Another alternative is to keep the noise parameters in comments scattered through the file, but Walksat doesn’t accept comments among the clauses, so the format would not have been backwardly compatible. Another option would be to put the noise parameters in the header, commented out for backward compatibility. This was discounted because the file is read sequentially. Files on standard input can’t be read in random order, and this means that walksat would not know how many clauses the instance has when reading in the noise parameters, thus allocating memory for them would be complicated. Currently this is done directly after reading the “p cnf ...” line. Memory allocation can be done statically, but this is wasteful and inflexible so I decided it should be dynamic. The generation of this format is covered in sections 3.3.2 and 3.4.

Walksat was altered in three ways:

1. The input reader was altered to accept the noise parameters, and store them in new data structures (arrays indexed by clause number).
2. The heuristics were altered to read the noise parameter from the arrays rather than use the global one.
3. Extra statistics printing was added as needed.

3.2 Post-processing

For the purpose of this project I used the median average of flips in a run, since this only requires half the runs to be successful and isn’t affected by outliers.

The output of Walksat contains a table by default, with various statistics about the runs it has performed. One of the statistics is the number of flips performed in a run. To obtain this, I wrote a parser program, which extracts the relevant data from the output of Walksat, sorts it and prints the median value. The “tries” command line option of walksat is used to control how many values are averaged. Unless stated otherwise, I used 101 throughout this report. The “cutoff” option is used to set the upper limit of the median; if it is set lower than the actual median, fewer than half the tries will succeed and the median won’t be found.
The parser program is capable of parsing a large number of output files from Walksat, and forming the data into a “comma separated values” file. In order to plot the data on a 3D chart, Matlab was used because it has good visualization facilities. Scripts (Matlab M-files) were written to automate importing the data and plotting it onto a chart. For the 2D charts, the StarOffice spreadsheet program was used.

3.3 Problem encoding

3.3.1 Col-NB

This utility was written by Timothy Peugniez to support the work of Frisch and Peugniez [1]. It encodes from a standard graph representation to an NB-SAT instance, with one parameter, the total colours, $C$. The resulting NB-SAT instance is satisfiable if and only if the graph is colourable in $C$ colours. The encoding is carried out as described in section 2.3.2. The graph format is the Dimacs standard format, shown here:

```plaintext
    c
    c Comment lines at start of file begin with 'c'
    c
    p edge <num_nodes> <num_arcs>
    e <node> <node>
    e <node> <node>
    ...
```

Each line beginning with ‘e’ represents an arc or edge on the graph. Nodes are represented by positive integers. This is encoded in the NB-CN format:

```plaintext
    c
    c Comment lines at start of file begin with 'c'
    c
    nb cnf <num_vars> <num_clauses> <domain_size>
    var_1{V1,V2,\ldots,Vi} var_2{V1,V2,\ldots,Vj} \ldots var_n{\ldots} 0
    ...
```

In this format, each line in the body represents one clause in the problem instance. Each line specifies a set of variables, each with a set of values enclosed in braces. For the clause to be satisfied, it is sufficient that one of the variables is assigned one of its corresponding values. A shortcut is also used in this file format; a literal of the form shown below is satisfied if and only if $var$ is not assigned any of $V1$...$Vi$. This makes the file shorter and more human-readable in the cases where only a handful of variables need to be specified with “!”, but without it a large number would be specified. It is also useful for encoding as CNF, because a variable with the “!” can be translated to negative literals in CNF.

```plaintext
    var!{V1,V2,\ldots,Vi}
```

The program takes its input on standard input and prints to standard output.
### Table 3.2: The command line arguments of NB-B2

<table>
<thead>
<tr>
<th>Argument</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>-atleastone</td>
<td>Include ALO clauses (default is to omit them)</td>
</tr>
<tr>
<td>-atmostone</td>
<td>Include AMO clauses (default is to omit them)</td>
</tr>
<tr>
<td>-negate</td>
<td>Translate negative ('!') non-Boolean literals to negative Boolean literals (default is positive)</td>
</tr>
<tr>
<td>- unary</td>
<td>Force unary translation in cases where domain size is 2 (it is default in all other cases)</td>
</tr>
<tr>
<td>- binary</td>
<td>Use the binary translation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Noise parameters:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-tabu</td>
<td>Specify that a tabu-style single noise parameter is used for each clause (default is numerator and denominator style)</td>
</tr>
<tr>
<td>-t&lt;type&gt; N</td>
<td>Specify that N is the tabu noise parameter for the clause type specified, type ∈ {kernel, alo, amo}</td>
</tr>
<tr>
<td>-p&lt;type&gt;&lt;nd&gt; N</td>
<td>Specify that N is the numerator or denominator for the clause type, type ∈ {kernel, alo, amo} and nd ∈ {num, den}</td>
</tr>
</tbody>
</table>

### 3.3.2 NB-B2

This utility encodes the NB-CNF file format described in the section above, into a CNF format with noise parameter annotation, described in section 3.1. It was modified from a tool written by Timothy Peugniez (NB-B), which does the same encoding but without the annotation. The command line options of NB-B2 are summarized in Table 3.2. It reads the input on standard input, and prints to standard output. The command syntax is quite complex, so I’ll illustrate it with an example, along with Col-NB. The command below takes a graph description and outputs a Boolean SAT instance, corresponding to the colouring instance on that graph with 16 colours. The encoding to SAT is unary, with negative kernel clauses and ALO clauses. The noise on kernel clauses is \( \frac{1}{10} \) and on ALO clauses it is \( \frac{3}{10} \).

```
Col-NB -cols 16 <input.col | NB-B2 -negate -atleastone -pkernelnum 15
-pkernelden 100 -palonum 35 -paloden 100 -pamonum 100 -pamoden 100 >output.cnf
```

One problem with this program is that even though AMO clauses are not required, the noise parameters for them must be specified anyway.

### 3.4 Problem instance generation with rrgen

The graph colouring instances used in this project were taken from a standard library. Round robin instances however were generated locally, using the rrgen program. This was written to output the annotated format described in section 3.1. Since I wrote it after most of the work described in section 4.2, I decided that it would not output the tabu-style noise annotation, just the style with numerator and denominator. Section 4.2 concludes that using multiple noise parameters for *tabu* would be very similar to *best*; essentially that behaviour on kernel
<table>
<thead>
<tr>
<th>Argument</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>-n N</td>
<td>Set the number of teams to N</td>
</tr>
<tr>
<td>-&lt;n&gt; Num Den</td>
<td>Set noise parameter for clause type n (numbered from 2-6 according to</td>
</tr>
<tr>
<td></td>
<td>the paper by Béjar and Manyà [10])</td>
</tr>
</tbody>
</table>

Table 3.3: The command line arguments of rrgen

clauses will not be affected significantly by the noise parameter, therefore experiments with round robin are very unlikely to produce any interesting results.

Rrgen directly generates round robin instances encoded as SAT (see section 2.3.3 for details of the encoding). It takes the arguments listed in Table 3.3, and it prints the encoded instance to standard output.
Chapter 4

Kernel noise irrelevance with *best* and *tabu*

In this chapter, I will show that the setting of a noise parameter for kernel clauses is irrelevant for *best*, and approximately so for *tabu*. This applies to the unary encoding, with graph colouring, round robin and other problems (if they are encoded in a similar way).

4.1 The *best* heuristic

When a kernel clause is picked by the Walksat algorithm, the result is that a node (in the original graph) has one colour removed. The reason a kernel clause would be unsatisfied is that the two Boolean variables in that clause are both *True* (since they are both negated). The figure below shows what happens to an individual node during the search. Since a random assignment assigns approximately half the variables to *True*, in this case the nodes will start with a mean of 12.5 colours each. Typically, there will be a large number of clashes at this point, and (assuming no nodes have 0 colours) kernel clauses will be picked exclusively. This causes a rapid reduction in the number of unsatisfied clauses early in the search.

This diagram shows the states of a graph node. The circles are labelled with the number of colours assigned to the node. Satisfying a kernel clause causes the rightward transitions, and satisfying an ALO clause causes the leftward ones. Since ALO clauses can only colour nodes with no colours, it isn't possible to colour a node that already has one or more colours.

4.1.1 Kernel noise irrelevance and a theorem to account for it

Where a kernel clause covers two nodes A and B, the table below shows all possible actions, none of which involve the noise setting.
<table>
<thead>
<tr>
<th>Colours in node A</th>
<th>Colours in node B</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;1</td>
<td>&gt;1</td>
<td>Random choice since neither action breaks any clauses</td>
</tr>
<tr>
<td>&gt;1</td>
<td>1</td>
<td>A is decoloured since this breaks no clauses</td>
</tr>
<tr>
<td>1</td>
<td>&gt;1</td>
<td>B is decoloured since this breaks no clauses</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Random choice since both actions break 1 ALO clause</td>
</tr>
</tbody>
</table>

In fact this theory can be generalized. Consider the case where the kernel clause has more than two variables; for example in the encoding of entirely negative NB-CNF to CNF with kernel and ALO clauses (so that the kernel clauses have length equal to the length of the original NB-CNF clauses). Now there are more than four cases, but the same pattern emerges. Where $n$ is the number of literals in a kernel clause, the table below shows the possible actions (again, none of which involve the noise parameter).

<table>
<thead>
<tr>
<th>Number of nodes with &gt;1 colours, represented in kernel clause</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Random choice since no variable breaks any clause</td>
</tr>
<tr>
<td>1..($n - 1$)</td>
<td>Random choice among those representing nodes with &gt;1 colours, since those variables break no clauses</td>
</tr>
<tr>
<td>$n$</td>
<td>Random choice because all variables break 1 ALO clause</td>
</tr>
</tbody>
</table>

So the theory holds for any number of variables in the kernel clauses. So far we have only considered the case where the kernel clauses are negative (i.e. all their variables are negated, and their action is to flip variables from true to false when selected). ALO clauses are positive, opposing the kernel clauses. The theory also holds when kernel clauses are positive and the opposing clauses are negative. An example of such a situation would be an encoding of graph colouring in which the kernel clauses express the constraint that two nodes connected by an arc must be coloured with different colours, as in Expression 4.1. Such an expression arises when the graph colouring to NB-SAT encoding is done as described in section 2.3.2 but the NB-SAT to SAT encoding (section 2.3.1) uses the positive form. The first of the three clauses in 4.1 prevents both Node1 and Node2 being green simultaneously, given that they have only one colour each. The second and third do the same for blue and red respectively. The opposing clauses required are AMO (section 2.3.1).

Where a node has 0 colours, colouring it breaks no clauses, and where a node has one colour, colouring it breaks the relevant AMO clause. Therefore if the first clause of Expression 4.1 were picked, and neither node were coloured, then the choice among the variables would be random. If one node were coloured then colouring the other would break no clauses, so it would be coloured red or blue. If both nodes were coloured, flipping any variable would break one clause (the corresponding AMO clause) so the tie is broken with a random choice. Again none of these cases use the noise factor $p$, and the theory holds.

\[(Node1/red \lor Node1/blue \lor Node2/red \lor Node2/blue) \land \]

\[(Node1/red \lor Node1/green \lor Node2/red \lor Node2/green) \land \]

\[(Node1/blue \lor Node1/green \lor Node2/blue \lor Node2/green) \quad (4.1)\]

A clause type that is unaffected by the noise setting has the following characteristics:
1. The literals in it all have the same polarity.

2. Flipping any variable in any clause of that type can only cause one or no other clauses to be broken.

For the theory to hold, the heuristic need not be best, but it must have certain characteristics:

1. The objective function must be breaks rather than makes−breaks, because the variables in a kernel clause may all break an opposing clause if flipped (and hence tie, all breaking one clause) but if makes are accounted for, the variables might not tie.

2. A tie must be broken with a random choice.

3. The heuristic must choose randomly among variables that break no clauses, if any such variables exist in the kernel clause chosen. This is so that variables that break no opposing clauses are chosen and the noise factor is ignored in that case.

4. The type of noisy move preferred by the heuristic must be random selection among all the variables in the clause.

4.1.2 How widely does this theorem apply?

Although the theorem above is described with reference to the encoded graph colouring problem, it also applies to the round robin problem encoding described in section 2.3.3. For clause types 2, 3, 4 and 5, the noise is irrelevant since they’re all negative, and flipping any variable in any of those clauses can only ever break one of the type 1 (ALO) clauses. The optimal noise parameter where only one is used is therefore equal to the optimal parameter for ALO clauses. This was found to be the most computationally competitive encoding by Béjar and Manyà [10], out of the six alternatives they tried. The unary graph colouring encoding with negative kernel clauses was also found to be fastest by Frisch and Peugniez [1] for four out of six example graphs. They tried three encodings.

It seems that unary encodings tend to have kernel clauses of one polarity and then ALO or AMO clauses. It is convenient to formulate a problem as SAT by disallowing all non-solutions by using a set of rules formulated as negative kernel clauses, then add ALO clauses to force an assignment to each variable in the problem instance. The result of this is that the theorem applies to a wide variety of encoded problems. Of course this doesn’t help pick a good noise setting, but it does at least discount the strategy of using more than one.

What about heuristics other than best? Even though the tabu heuristic doesn’t meet all the requirements, it does approximately conform to the theory. Novelty and rnovelty don’t meet any of the requirements, and don’t conform at all.

4.2 The tabu heuristic

Tabu is similar to best in that it uses breaks as its objective function but the mechanism for introducing noisy moves is significantly different. A list is kept of the variables flipped in the previous t steps; tabu refuses to flip a variable that is in the list, with the exception of variables that have no breaks. The heuristic first chooses among those variables with no breaks. If there are no such variables, it chooses the variable with fewest breaks among those not in the tabu
list. If there are none to choose among, another clause is picked. If no unsatisfied clause has a variable that is not on the tabu list, the list is ignored.

I have adapted tabu for multiple noise parameters in the following way: where the clause has an associated list length of \( t \), the algorithm uses the history of the search over the previous \( t \) steps as the tabu list. In this way, clauses with the same tabu list length share the same tabu list, and a clause with a tabu list length that is greater than \( t \) will share the same tabu list but with some extra entries. This isn’t the only way, another option would be to keep a private tabu list for each clause type, containing only the history of moves for that type. I believe this second approach is inconsistent with the original tabu because it changes the behaviour across clause types. For example, for clause types \( A \) and \( B \): if a variable is first flipped in a clause of type \( A \), and soon after (within the length of \( B \)’s tabu list) is considered in a clause of type \( B \), it will not be on \( B \)’s tabu list whereas in the unaltered heuristic (and my favoured alteration) it would be.

4.2.1 Is kernel noise irrelevant?

For the kernel noise parameter to be irrelevant according to the theorem in section 4.1, tabu should conform to the four conditions listed there, but in fact it only conforms to the first and the third since its mechanism for introducing noise is different. Figure 4.1 is a plot of median flips against kernel and ALO noise settings, for an instance of graph colouring encoding according to sections 2.3.2 and 2.3.1. Note for all charts of this type, there is a cutoff of 100,000 median flips, which shows on the chart as a flat plane. The depression on the chart runs approximately parallel to the kernel noise axis. The kernel noise parameter is largely irrelevant for this instance (although now there is no guarantee that it will be for all instances of graph colouring).

The fact that it doesn’t have the required noise mechanism seems to introduce a feature to the data: the trough running parallel to the kernel noise axis is narrower at low values of kernel noise. The base of the trough is of roughly equal depth all along (the lowest point where kernel parameter=20 is 19156 flips, where kernel parameter=10 it is 16622 and where kernel parameter=0 it is 17690). The deep part of the trough, represented in blue on the chart, gets wider as kernel noise decreases, but the areas represented in green, yellow and red become steeper. The combination of these two effects is that the trough is narrower.

Figures 4.2 and 4.3 also show interesting behaviour where kernel noise is low (note that Figure 4.3 is viewed from a different angle than the other two). Tabu behaves identically to best when both have their noise parameters set to zero, because the difference is in the noise mechanism and setting the parameter to zero essentially disables this mechanism. As the kernel noise parameter is increased from 0, the behaviour of tabu on kernel clauses diverges from that of best.

Despite the behaviour at low kernel noise, the kernel noise parameter is largely irrelevant for all three instances.

4.2.2 The behaviour at low kernel noise

Variables on the tabu list that are also in the chosen kernel clause must have been recently flipped by an ALO clause, since ALO clauses are the only positive ones. Their corresponding node on the original graph was recently coloured with one colour. Table 4.1 enumerates all possibilities for a kernel clause where both variables break one clause (this is the only case
Figure 4.1: Median flips, instance le450_25a, coloured with 25 colours, *tabu*

Figure 4.2: Median flips, instance le450_15a, coloured with 16 colours, *tabu*
Figure 4.3: Median flips, instance le450_5d, coloured with 5 colours, \( tabu \)

<table>
<thead>
<tr>
<th>Number of variables from clause in tabu list</th>
<th>Action</th>
<th>Conformance to theorem (section 4.1.1)</th>
<th>Reason for non-conformance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Chooses randomly between the two variables</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Chooses the variable not in the tabu list</td>
<td>No</td>
<td>Choice between variables is influenced by recent flips</td>
</tr>
<tr>
<td>2</td>
<td>Pick another unsatisfied clause</td>
<td>No</td>
<td>Choice excludes both variables</td>
</tr>
</tbody>
</table>

Table 4.1: Actions of \( tabu \) on kernel clauses where both variables break one clause
in which the tabu list is involved, the case where one or both variables break no clauses is identical to best, and each variable in a kernel clause cannot break more than one clause.

However the tabu list is constructed, having more variables in there would increase the probability of a randomly-chosen variable being present. That probability is \( \frac{t}{N} \) where \( t \) is the length of the list and \( N \) is the total number of variables. Increasing the tabu list length by a constant amount has a decreasing effect on that probability as the tabu list size increases. If we assume that the variables considered for flipping by Walksat are random (when in fact they are chosen by a procedure that is partially random), we can link this to the behaviour of tabu on kernel clauses. In Figures 4.1 and 4.2, the trough tends towards being parallel to the kernel noise axis, as the kernel noise parameter (tabu list length used for kernel clauses) is increased. Figure 4.3 shows that the edge of the basin tends to become parallel to the kernel noise axis as the kernel noise parameter is increased.

To test this theory, normally a log scale chart might be drawn, of the optimal ALO noise against kernel noise. Unfortunately, because the tabu list takes a discrete length, it is impossible for any of the three instances used here.

The tabu list at some time \( t \) during the search is not a complete unknown, because the moves leading up to \( x \) influence both the move made at \( x \) and the tabu list. In the early stages of the search, I would expect large numbers of unsatisfied clauses, covering between them the majority of the variables. Since the clause is chosen with uniform probability, the search will jump about, and the tabu list will be nearly random. However, as the search progresses, I would expect the tabu list to become more closely related to the variables in the clause chosen at time \( t \), because the search will probably be in a local minimum (near-solution point) in the search space, and only a small number of related clauses will be unsatisfied (for example, where all the unsatisfied clauses relate to one tightly constrained area of a graph). Despite this relationship, the contents of the tabu list are random enough to allow the property described above.

### 4.2.3 Difficulty and the noise parameters

Figure 4.3 has a trough that is wider than the other two, it extends off the chart. This suggests that the le450.5d instance is considerably easier for tabu to solve than either of the others, since in this case it appears that the width of the trough is influenced by problem difficulty at the optimum setting of the noise parameters. The lowest point of Figure 4.2 has 28116 median flips, and the trough is narrow. For Figure 4.1, the lowest point is 14785 and the trough is wider than that of Figure 4.2. Figure 4.3 has the widest trough and a lowest point of merely 8619 flips. However, ‘the steepness of the curve [of median flips plotted against the noise parameter] is not purely a function of the hardness of the problem [instance] at its optimal noise setting’ [7] so it is dangerous to apply this idea too freely.

### 4.2.4 Conclusions

All three of the problem instances considered cannot be sped up significantly by adjusting kernel and ALO noise independently, since the trough or basin extends over the line where kernel noise equals ALO noise. In all three the trough extends up to the line where kernel noise equals zero. Along with the fact that the troughs run approximately parallel to the kernel noise axis, this convinces me that all graph colouring instances encoded in this way cannot be improved by using multiple noise parameters. I conjecture that this conclusion extends to
other problems, when they are encoded in a unary way with kernel clauses of a single polarity. For this reason, I haven’t investigated the round robin problem with tabu.

Tabu would in my opinion also benefit from the new initializations considered in section 6.1 (although they are only tested with the best heuristic).
Chapter 5

An investigation of \((r)\text{novelty}\)

This chapter looks at \((r)\text{novelty}\) with multiple noise parameters, and finds that unary encoded graph colouring can be solved faster with multiple noise parameters. Unfortunately, this does not extend to unary encoded round robin.

5.1 The novelty heuristic

5.1.1 Graph colouring

Encoding to SAT was carried out as in sections 2.3.1 and 2.3.2. Figure 5.1 shows the median flips of the le450_25a instance, which novelty finds easy in comparison to two others considered later. The chart shows that a wide range of noise settings are good for this instance, with the exception of low noise for either ALO or kernel clauses. Low noise on both clause types appears to damage performance more than on just one type. The cross-section of Figure 5.1 where the two noise parameters are equal is shown in Figure 5.2, and shows that there is a small rise in median flips at noise levels higher than 0.7. Although at first glance it would appear that high noise is good for graph colouring (encoded as it is), Figures 5.3 and 5.4 tell a different story. The small rise in Figure 5.2 is amplified for these other instances. le450_15a and le450_5d proved harder for best to solve at its optimal noise level, and this appears to have been reflected in the narrower trough that shows good performance with novelty. Note that a cutoff of 100,000 flips was used, resulting in planes on the 3D charts at that value, where the actual median flips value would be higher.

Table 5.1 shows that there is some modest advantage to varying kernel and ALO noise independently (at least for le450_25a and le450_15a, the difference with le450_5d is very small and could be due to statistical noise). Comparing novelty with best in Table 5.1, novelty shows itself to be better at handling the graph colouring instances. For example, the colouring of le450_5d takes 22,976 median flips with best, and 9,553 median flips with novelty, both using the optimal single noise parameter. Novelty takes just over 4% of best’s median flips!

Interestingly, for instances le450_25a and le450_15a the best median flips value occurs where kernel noise is set to 1, with ALO noise set to 0.4 for le450_25a and 0.15 for le450_15a. For le450_5d, the best result with kernel noise set to 1 was 10,698 flips, 16% higher than the overall best result. There may be some relation to total colours here, le450_5d was coloured with fewer than either of the others. Overall, novelty offers little promise of dramatically improving the speed of the search by using multiple noise parameters, because the troughs
Figure 5.1: Median flips, graph le450_25a, coloured with 25 colours

Figure 5.2: Equal noise cross-section of Figure 5.1

<table>
<thead>
<tr>
<th>Graph</th>
<th>Best median flips with single noise parameter</th>
<th>Best median flips with two noise parameters</th>
<th>Percentage improvement by using two parameters</th>
<th>Lowest median flips achieved by best</th>
</tr>
</thead>
<tbody>
<tr>
<td>le450_25a</td>
<td>18601</td>
<td>15403</td>
<td>17%</td>
<td>11519</td>
</tr>
<tr>
<td>le450_15a</td>
<td>19134</td>
<td>17306</td>
<td>10%</td>
<td>44731</td>
</tr>
<tr>
<td>le450_5d</td>
<td>9553</td>
<td>9224</td>
<td>3%</td>
<td>229762</td>
</tr>
</tbody>
</table>

Table 5.1: Results for novelty (and comparison with best)
in Figures 5.1, 5.3 and 5.4 all cross the line where kernel noise equals ALO noise. The three charts have a similar shape, a trough curved around the (1,1) point.

If one noise parameter is increased, the optimum point of the other decreases on all three instances. This interaction between the two noise parameters suggests to me that there is some sort of total noise, which has some optimal level for each problem instance. However, this doesn’t explain the magnitude of the effect: changing the ALO noise parameter (on all 3 instances for which we have data) from 1 to 0.8 has much less effect on the optimum point of the kernel noise parameter than changing it from 0.8 to 0.6. In other words, the trough is curved when a total noise theory would require it to be an approximately straight line. So why is the trough curved?

There are two cases in which a noisy move occurs (\(p_{alo}\) and \(p_{kernel}\) are the two noise parameters):

1. A variable \(v\) is flipped from true to false by a kernel clause, and subsequently the ALO clause containing \(v\) is selected, and \(v\) is the best variable. With probability \(p_{alo}\), another variable \(w\) is picked, and its corresponding colour is added to the node which both \(v\) and \(w\) represent.

2. A variable \(v\) is flipped from false to true by an ALO clause, and then a kernel clause containing \(v\) is selected, and \(v\) is the best variable. With probability \(p_{kernel}\), another variable is picked, representing the same colour as \(v\) for another node, and that second node is uncoloured instead.
These two cases are fairly different, in that case 1 involves only one node of the graph, and case 2 involves two. Also case 1 has many other variables in the second clause, any of which could be picked, whereas in case 2 there is only one other variable in the clause, but potentially a large number of clauses that could be picked. The reason for the curved troughs must therefore be quite general to extend to both of these cases.

5.1.2 Better performance where kernel noise is 1

Instances le450_25a and le450_15a show an interesting characteristic: the lowest value for median flips occurs where kernel noise is 1, and ALO noise is a low value (0.4 and 0.15 respectively). For le450_15a, the median flips at that point is 10% lower than where the two noise parameters are equal, and 38% lower than where ALO noise is 1 and kernel noise is 0.2 (the optimal where ALO noise is 1). So why is this? I could think of two possible explanations. The first is below:

For novelty, the noise parameter is involved whenever the best variable under the makes–breaks sort is also the most recently flipped. For a kernel clause, if the best variable is $p^n_c$ where $n$ is the node and $c$ is the colour (since each variable is related to a unique pair of node and colour), and $p^n_c$ is the most recently flipped, then the previous move was the colouring of node $n$ with colour $c$. For an ALO clause, if variable $p^n_c$ is the best variable and it is the most recently flipped, this means that the previous move was the removal of colour $c$ from node $n$ by a kernel clause. A noisy move under these conditions is not to choose $p^n_c$ but the second-best variable. For a kernel clause, the second-best variable represents the same colour.

Figure 5.4: Median flips, graph le450 _5d, coloured with 5 colours
but a different node, and for an ALO clause the second-best variable represents the same node and a different colour.

The observation above suggests that it is better for noisy moves to be made by kernel clauses than ALO. This suggests that the objective function is effective for ALO clauses. The colour chosen by the ALO clause is good, and the node should not be immediately uncoloured by a kernel clause. If a noisy move is required to escape a local minimum, it is better to uncolour the other node represented in the kernel clause. In effect, this moves the search to another node, because that other node now needs to be coloured (assuming it had only one colour previously). This should be good for the mobility or coverage (or both) of the search. The data is showing that it is better to move around the graph rapidly, greedily satisfying constraints as you go.

A second possible explanation of this observation is that kernel clauses are only two literals long, and on 2-SAT (the subset of SAT where all clauses are two literals long) Walksat with the best heuristic performs optimally when the noise parameter is set to 1 [7]. In other words, on such instances the objective function is a hindrance and should always be ignored. This is using a different heuristic, and only the kernel clauses are two literals long, and the variables are shared with clauses of a different length. Therefore I don’t think this is a likely explanation.

This means multiple noise parameters can allow better performance than one for graph colouring. However, the improvement is small, 17% for instance le450_25a (25 total colours) and 10% for le450_15a (16 total colours). Instance le450_5d (5 total colours) offers no improvement. There may be some relationship to total colours here, over these three instances, fewer total colours has resulted in a smaller improvement. Clearly, three instances is not a large enough data set to say this for certain.

5.1.3 An alternative encoding for graph colouring

Section 2.3.4 justifies the use of the unary encoding, and I have been using the negative unary encoding shown to be efficient by Frisch and Puegniez [1]. There is another alternative however, a unary encoding where the kernel clauses are positive, described in section 2.3.1. The positive encoding of the kernel clauses assumes that each node can have only one colour, otherwise constraints on the colours are not maintained. Therefore AMO clauses are required. ALO clauses are not required because the kernel clauses ensure that the nodes have at least one colour each.

The positive encoding is generally thought of as worse than the negative, because it produces large SAT instances in terms of both clauses and literals per clause, and it takes more flips in general to reach a solution. It is possible that the reason for the sub-optimality of this encoding is that it needs multiple parameters to perform well (i.e. to optimize depth, mobility and coverage with one clause type causes them to become significantly suboptimal with another, and the overall result is that with only one parameter, setting it for good performance is impossible).

I tested instance le450_15a with all combinations of kernel and AMO noise settings, where each is in the range 0,1 and is increased in steps of 0.05. The cutoff was set to 200000 (double that used for the results shown in Figure 5.3). Unfortunately, no combination of noise settings yielded good performance, or even performance good enough to calculate the median.

In conclusion, the positive encoding is not poor because the clause types require different noise settings, but for some other reason, possibly that it produces an unnecessarily large number of clauses.

38
5.1.4 The round robin problem

Figure 5.5 shows that if a single noise parameter is used with the round robin problem instance where there are 10 teams (with the unary encoding to SAT as described in section 2.3.3) the optimum noise parameter is 0.18. This instance is quite sensitive to the noise parameter being set sub-optimally.

Figure 5.6 shows the effects of varying each of the five noise parameters of the round robin instance. For this experiment, the noise parameters that are not varied are set to 0.18 (the optimum where one noise parameter is used). The number of medians to be found is thus limited to 5x when there are x⁵ if all combinations of noise parameters were tried, where x is the number of settings used for each noise parameter. In this case x = 21, so the interval between two noise settings is 0.05.

The first thing to be noted is that most of the charts are spiky, an effect normally attributed to an inadequate number of samples. Chart (a) shows the type of curve I would have expected, a fairly smooth curve. Chart (c) is also quite regular. Charts (b), (d) and (e) are erratic but note that the y-axes have different scales for each chart, so they’re not so bad as they first appear. Each chart in Figure 5.6 was plotted from 201 samples at each noise setting. To confirm that the spikiness is an effect of the sampling, not some characteristic of the variables being observed, Figure 5.7 shows another sample of 201 values, for the three most erratic charts. As you can see, the overall shape is the same but the spikes are different. The fact that the average tends to be erratic is a symptom of a lot of variation in the number of flips.

Another thing to note about these charts is that the kernel parameters all have a similar effect on median flips, with varying degrees of spikiness and varying steepness. Their optima are approximately equal to the single noise parameter optimum. Given this similarity, I speculate that restricting the kernel noise parameters to be equal will not damage performance.
Figure 5.6: Varying a noise parameter while others are constant, *novelty*
Figure 5.7: A second sample of some charts in Figure 5.6
5.1.5 Comparison of round robin to graph colouring

Figure 5.6 shows that each noise parameter is much less sensitive to a sub-optimal setting than the single noise parameter shown in Figure 5.5. This is to be expected, changing the noise for one set of clauses isn’t likely to have as much effect as changing it for all. Figure 5.8 contains another plot of the data collected for the le450_15a graph colouring instance, in a format that shows clearly that the single noise parameter has a much steeper chart than either of the others. This shows that changing the noise for one set of clauses also has a diminished effect with graph colouring.

The optimum of each chart in Figure 5.6 is approximately equal to the optimum of the single noise parameter. This also holds for graph colouring, and can be seen clearly for graph le450_15a in Figure 5.8.

With graph colouring the noise parameters interact with each other, and (given the similarities of encoding) it is likely that the same holds for round robin. In section 5.1.2, it was noted that graph colouring performed better where kernel noise is 1, so perhaps round robin will perform better with high kernel noise.

5.1.6 Round robin with high kernel noise

Figure 5.9 shows what happens if all the round robin kernel parameters are varied together. This gives two variables, ALO noise and kernel noise. The chart shows that no improvement is possible by setting all the kernel noise parameters high, for any value of ALO noise. This chart shows a trough that is shaped a lot like the analogous trough for a graph colouring instance, for example Figure 5.3 on page 36, but with the difference that the trough is restricted by the range of the noise parameters. Unfortunately, it seems that ALO noise cannot be low enough to take advantage of high kernel noise, and as a result the overall optimum is close to the line where all the noise parameters are equal (the deviation from that line is probably insignificant).

Figure 5.9 was plotted with an interval between noise settings of 0.05, so it is feasible that a better setting lies between 0.05 and 0 ALO noise, where kernel noise is 1. Doing some further experimentation revealed that the optimum ALO noise is approximately 0.02, and that results between 24 000 and 30 000 median flips can be achieved between 0.008 to 0.04 ALO noise. However, this is still some way short of the overall optimum of 11 324 (where the two noise
Figure 5.9: Varying kernel noise parameters together, *novelty*
The overall optimum occurs where both parameters are set low, and the extent of the trough is limited by the lower bound of the noise parameters. This suggests to me that a stronger objective function is required. *makes – breaks* can be augmented by considering the strengthening of clauses (a clause is stronger if more of its literals are true). The SDF local search algorithm invented by Schuurmans and Southington [8] chooses variables directly. It finds the best under the *makes – breaks* sort, then breaks ties by looking at the clauses they strengthen. SDF competes well with Walksat when comparing flips, but can lag when comparing CPU time, primarily because the current best implementation of SDF uses floating-point arithmetic in its main loop [8]. SDF does not take a noise parameter, and doesn’t make noisy moves. Instead the objective function is dynamically adjusted to find a good way out of a local minimum, and that objective function nearly never ties (when it does, the tie is broken with a random choice). ‘SDF is competitive with the previous systems (WSAT [meaning Walksat with *best*], Novelty+ [a modified *novelty*]) in CPU time on large structured problems’ [8, page 25]. Unary encoded round robin is a structured problem (and can be large, depending on the number of teams).

Béjar and Manyà used *rnovelty* successfully on unary encoded round robin, and found that the optimal single noise parameter reduces as the instances grow in size and difficulty [10]. Figure 5.10 shows that the larger instance where \( n = 12 \) has a lower optimal single noise parameter than \( n = 10 \), for *novelty*. The optimum setting for \( n = 10 \) is 0.18, whereas for \( n = 12 \) it is 0.15. Given this result, and the fact that *rnovelty* and *novelty* are similar, it is likely that the optimal noise parameter for *novelty* also declines as the instances grow in size and difficulty. This is evidence supporting my conjecture that a stronger objective function is required, particularly for larger instances.

The equivalent of Figure 5.9 for higher values of \( n \) is likely to be much the same, but with an overall optimum that is closer to the \((0,0)\) point. As a result, I think that using an ALO and one kernel noise parameter for round robin will not allow an improvement in performance at any \( n \geq 10 \).
What if the kernel parameters are actually significantly different, despite the observation in section 5.1.4? Experiments showed that with one kernel parameter set to 1, and all four other noise parameters varied together, performance worsened relative to all five noise parameters being equal (for instance \( n = 10 \)).

5.1.7 Round robin conclusion

This section has covered some possibilities for setting the five noise parameters of round robin where \( n = 10 \), but no improvement has been achieved over using a single noise parameter. From the possibilities I have tried, it seems likely that multiple noise parameters offer no improvement for the instance where \( n = 10 \), or in fact any instance where \( n \geq 10 \). From Figure 5.6 on page 40, the kernel noise parameters all have a similar effect (with varying magnitude), so it is logical to vary them together. The data presented in Figure 5.9 shows that when kernel noise parameters are varied together, no improvement is made over a single noise parameter. Setting individual kernel parameters to 1 also failed to improve performance.

5.2 The \textit{r}novelty heuristic

5.2.1 Graph colouring

Three experiments were carried out on the same instances of graph colouring as were used with \textit{novelty}, again using the unary encoding. \textit{Rnovelty} is similar to \textit{novelty} in that they both use makes – breaks as the objective function, and they both use the best variable under that sort unless it was the most recently flipped, in which case they make a choice between the best and second-best variables (the nature of the choice is the only way in which they differ). For this reason, I will be referring back to the previous section when discussing \textit{r}novelty.

Experimental results on instance \( le450 \_25a \) are shown in Figure 5.11. The chart shows clearly that the search performs well where kernel noise is greater than 0.5, and ALO noise is not 0. The variations in the solution time in the low part of the chart are small compared to the boundary where kernel noise is 0.5. Even so, it can be seen that the optimal setting of the noise parameters is well off the line where kernel noise equals ALO noise. It is where kernel noise is 1 and ALO noise is 0.2. To compare this to Figure 5.1 on page 35 (the chart for the same instance run with \textit{novelty}), they are similar where kernel noise is greater than 0.5.

Figure 5.12 is a much more interesting result. If a single noise parameter is used with this instance, performance is terrible but if two are used there is a “notch” where performance is good. The green line indicates where the noise parameters are equal. Figure 5.12 is similar to the chart for the same problem instance, with the \textit{novelty} heuristic, Figure 5.3 on page 36, but with the same characteristic noted with instance \( le450 \_25a \), that performance is awful below 0.5 kernel noise. From this it seems likely that any instance that requires only a small amount of noise (with \textit{(r)novelty}) will perform badly with \textit{r}novelty when a single noise parameter is used. This is the only instance seen that shows significant improvement from using multiple noise parameters. It is unfortunate that this improvement is due to a performance problem with a single noise parameter, rather than a great leap forward from using two.

A third example, Figure 5.13, shows that the chart for \textit{r}novelty is not always similar to the chart for \textit{novelty} run on the same problem instance. The trough of Figure 5.13 curves more closely to the (1,1) point than the equivalent chart for \textit{novelty} (Figure 5.4 on page 37), and the trough is much narrower for \textit{r}novelty as well. Because the trough is closer to the (1,1)
Figure 5.11: Median flips, graph le450_25a, coloured with 25 colours, \textit{rnovelty}

Figure 5.12: Median flips, graph le450_15a, coloured with 16 colours, \textit{rnovelty}
point, it avoids the performance problem below 0.5 kernel noise. For an explanation of why performance is awful below 0.5, please see section 6.3, where the problem is identified, and a modified \textit{novelty} is demonstrated without this problem.

It is hard to tell on these charts whether the property noted in section 5.1.2 (for \textit{novelty}) is also true for \textit{rnovelty}. Figure 5.11 shows a slope down towards the point where kernel noise is 1 and ALO noise is 0.2, which suggests that the property does hold for \textit{rnovelty}. This could be good news, since \textit{rnovelty} is often the fastest heuristic in terms of flips [10, 2].

Table 5.2 summarizes the performance of \textit{novelty} and \textit{rnovelty}, with single and multiple noise parameters. \textit{Rnovelty} shows a better improvement than \textit{novelty} for all three instances, but the figures for a single noise parameter are worse than \textit{novelty}. The charts for instance le450_5d suggest that the improvement shown for that is probably not valid, since the charts show troughs that are approximately the same depth along their length, with some variation that is probably due to sampling. Where the best median flips value is starred, it occurred where kernel noise=1.

\textbf{5.2.2 The round robin problem}

The round robin instance where $n = 10$ and with a single noise parameter has an optimal noise level of 0.1. This is slightly lower than 0.18 for \textit{novelty}. With \textit{rnovelty}, this instance is even more sensitive to sub-optimal noise than with \textit{novelty}, as you can see from Figure 5.14. The best performances of the two heuristics are very similar. In this experiment, I used the unary encoding to SAT as described in section 2.3.3, which is similar to the encoding used for graph colouring. However, the poor performance of graph colouring below 0.5 kernel noise has
<table>
<thead>
<tr>
<th>Graph</th>
<th>Heuristic</th>
<th>Best median flips with single noise parameter</th>
<th>Best median flips with two noise parameters</th>
<th>Percentage improvement</th>
<th>Lowest median flips achieved by best</th>
</tr>
</thead>
<tbody>
<tr>
<td>le450_25a</td>
<td>novelty</td>
<td>18601</td>
<td>15403*</td>
<td>17%</td>
<td>11519</td>
</tr>
<tr>
<td></td>
<td>rnovelty</td>
<td>21015</td>
<td>13170*</td>
<td>37%</td>
<td></td>
</tr>
<tr>
<td>le450_15a</td>
<td>novelty</td>
<td>19134</td>
<td>17306*</td>
<td>10%</td>
<td>44731</td>
</tr>
<tr>
<td></td>
<td>rnovelty</td>
<td>&gt;100 000</td>
<td>17861</td>
<td>&gt;82%</td>
<td></td>
</tr>
<tr>
<td>le450_5d</td>
<td>novelty</td>
<td>9553</td>
<td>9224</td>
<td>3%</td>
<td>229762</td>
</tr>
<tr>
<td></td>
<td>rnovelty</td>
<td>10353</td>
<td>7433</td>
<td>28%</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2: Results for novelty and rnovelty

![Single Noise Parameter Graph](image)

Figure 5.14: Round robin where $n = 10$, single noise parameter

not carried over at all to round robin, indeed the optimal noise setting is well below 0.5. This is good news, because it proves that unary encodings can be used effectively with standard rnovelty.

Figure 5.15 shows the effects of varying each of the five noise noise parameters of the round robin problem instance. For this experiment, the noise parameters that are not varied are set to 0.1 (the optimum where one noise parameter is used). Similarly to novelty, no noise parameter when varied individually has as much effect as varying them together. Chart (a) is noticeably steeper than the equivalent chart for novelty, but this is to be expected because the chart for a single noise parameter is also steeper than the equivalent for novelty. Charts (b), (c), (d) and (e) are similar to the equivalent charts for novelty.

Overall, the results are similar to those for novelty, and I expect that the same conclusions (in section 5.1.7, page 45) will hold. Hence I expect no improvement in speed can be made by using multiple noise parameters with round robin and rnovelty.
Figure 5.15: Varying a noise parameter while others are constant, *novelty*.
Chapter 6

Future Directions

This chapter looks at ways in which this work may be expanded. It discusses some possibilities for specializing Walksat to specific problems, and looks at possible other ways of dividing the clauses of a SAT instance, unrelated to the encoding of a problem.

6.1 Tuning best to graph colouring

6.1.1 Faster processing of kernel clauses

Since kernel noise has no effect at all when using best (section 4.1.1), no time saving can be made by setting the two noise parameters independently. However, Best could be tuned specifically to the graph colouring problem as a result of this finding. Kernel clause tagging could be used to improve performance when a kernel clause is selected by Walksat. Currently, when both nodes represented in the kernel clause have one colour, with probability $p$ the program performs a random choice between the variables. With probability $1 - p$ the program compares the number of clauses broken by each variable then performs a random choice (because they break an equal number of clauses). This fruitless work could be replaced by a simple random choice, probably yielding a modest linear performance gain. This alteration would also work for round robin, with clause types 2 to 5 (listed in section 2.3.3) labelled as kernel clauses. The proportion of kernel clauses and ALO clauses picked during the search will be roughly even because they oppose each other, so the proportion of kernel clauses to ALO clauses will have no effect.

6.1.2 New initializations

The rapid descent performed early in the search could also be removed, because choices between variables during the descent are entirely random (given both variables involved in the choice have >1 colours). Instead, an assignment giving each node one random colour would remove the pointless work, or (possibly even better) the program could avoid the work of picking an initial assignment as well, and the search could start with all variables set to false (i.e. all nodes uncoloured). The first possibility is similar to the initial state of a local search procedure that operates directly on graph colouring. However, it differs from a direct local search in that partial assignments are allowed (this could be an advantage in that the partial assignments are useful as intermediate states).
<table>
<thead>
<tr>
<th>Graph</th>
<th>Approx. optimal noise</th>
<th>Median flips with uncoloured initialisation</th>
<th>Median flips with one colour per node initialisation</th>
<th>Median flips with normal Walksat initialisation</th>
<th>Total colours</th>
<th>( \frac{(\text{TotalColours} - 1)}{2} \times \text{Nodes} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>le450_25a</td>
<td>0.2</td>
<td>4024</td>
<td>4173</td>
<td>11519</td>
<td>25</td>
<td>5179</td>
</tr>
<tr>
<td>le450_15a</td>
<td>0.2</td>
<td>39156</td>
<td>40256</td>
<td>44731</td>
<td>16</td>
<td>3600</td>
</tr>
<tr>
<td>le450_5d</td>
<td>0.65</td>
<td>211938</td>
<td>211702</td>
<td>229762</td>
<td>5</td>
<td>1125</td>
</tr>
</tbody>
</table>

Table 6.1: Results for colouring instances, with different initializations

The second possibility (starting with all nodes uncoloured) is similar to the starting point of a systematic search procedure operating directly on the graph colouring problem, but of course the search is substantially different. In my intuition, starting with all nodes uncoloured would be better, since this would avoid starting from an assignment with many conflicts. The search would start to colour nodes (ALO clauses would be chosen) and colourings would initially avoid causing a conflict since the best heuristic always chooses a variable that breaks no clauses, if one exists.

To compare the two possibilities, experiments were performed on three instances of graph colouring with varied numbers of colours and difficulty. The results are in Table 6.1 and show an overall improvement over the previous best average, particularly where the total colours is high. This is because \( \frac{(\text{TotalColours} - 1)}{2} \times \text{Nodes} \) flips are saved in the early part of the search, since a node would average \( \frac{\text{TotalColours}}{2} \) colours after a random initialization, and (unless the instance is very easy) will not stand much chance of satisfying all the relevant kernel clauses until it has only 1 colour. Therefore \( \frac{(\text{TotalColours} - 1)}{2} \) moves can be saved per node.

Savings were made in excess of \( \frac{(\text{TotalColours} - 1)}{2} \times \text{Nodes} \) flips though, for both new initialization types. In an effort to find out why, I ran Walksat on the le450_25a instance and printed out the assignment after 5179 flips. The mean number of colours per node was 2.20, which tells us that the search is making a significant number of moves that don’t uncolour nodes, before the 5179th flip. Recall that kernel clauses uncolour a node when chosen and ALO clauses colour a node but only when that node has no colours; the search must be uncolouring some nodes to 0 colours then colouring them again, presumably in the most tightly constrained areas of the graph. From Figure 6.1, the number of unsatisfied clauses on chart (a) (which is a successful run with the unaltered initialization) after 5179 flips is \( 10^{1.5} \), considerably lower than charts (b) and (c) (which are successful runs with the two new initializations) which are both in excess of \( 10^{2.5} \) unsatisfied clauses at 0 flips. It seems that the work done before 5179 flips does reduce the number of unsatisfied clauses, but the fact that some nodes have more than one colour at 5179 flips hinders the search somewhat from that point on.

Table 6.2 shows the number of colours assigned to the first 10 nodes (the instance has 450). The table shows that for the most part, nodes have one colour each after 5179 flips. However, there are exceptions like node 6. Solutions can contain nodes with more than one colour, the extra colours simply indicate other possible graph colourings that would meet the arc constraints (for example, if in the Boolean assignment, one node had two colours and all others had one, that would map to two solutions in graph colouring). If the search space
Figure 6.1: Comparison of unsatisfied clauses during run
includes assignments with more than one colour per node, solutions are likely to be much sparser than in a search space that only allows one or no colours per node. Sparsity is a potential advantage [2]. However, intuitively, arc constraints are easier to meet when the number of colours per node can be no more than one at any point in the search space, which clearly is the case for the new initializations we are considering. This would explain the time saving in excess of \((\text{TotalColours} - 1) \times \text{Nodes}\) flips.

In Table 6.1 the two easier instances (le450_25a 25-coloured and le450_15a 16-coloured) get slightly better results with the uncoloured initialization, and the harder instance (le450_5d, 5-coloured) is solved slightly faster with the one colour per node initialization. Clearly the advantage offered by the new initializations is highly dependent on the total number of colours, but it also depends on the difficulty of the instance since hard instances have a long tail, causing the time saved in the early part of the search to become insignificant.

Figure 6.1 shows a single successful run of Walksat on le450_25a for each initialization, with noise set to 0.2. The number of unsatisfied clauses was sampled every 100 flips. (a) is a run with the normal initialization, (b) is one with the all false initialization and (c) is one with the one colour per node initialization. Note that the scales are different. The difference in length between (b) and (c) is irrelevant; the averaged figures show a much smaller difference in total flips than the charts. What is interesting about (b) and (c) however is their similarity. Starting with two apparently very different initializations, (b) and (c) have a similar starting point and early shape. In contrast, chart (a) takes approximately 4000 flips to reach \(10^{2.75}\) unsatisfied clauses, but from that point on, is of a similar shape to (b) and (c), as expected. Chart (a) has a depth (after the early rapid descent) of approximately 7 (calculated by taking the mean of the samples after 6500 flips), whereas charts (b) and (c) have depths of about 5.9 and 5.3 respectively (taking the mean from 800 flips onwards). This higher value for (a) could be caused by some nodes remaining multicoloured after 6500 flips, causing more conflicts than would otherwise be the case.

### 6.1.3 Conclusions about new initializations

The new initializations considered give a gain related not to the median flips, but to a function of total colours and nodes in the graph. Large, easy problem instances therefore get the best effect. This report does not cover the use of these new initializations on the other heuristics,
but I speculate that the results found here apply to all of them, and indeed to other local search procedures. In my intuition, starting the search at a position closer to the solution must be good for average performance. New initializations of the same type would be very likely to benefit other unary SAT-encoded problems in a similar way. Round robin could benefit from starting with just one team assigned to each slot, for example. I believe this has potential for a wide range of problems.

6.2 Tuning tabu to graph colouring

It was shown in section 4.2 that, for unary SAT-encoded graph colouring, the kernel noise parameter is unimportant. The tabu heuristic does quite a large amount of work for the kernel clauses, which is mostly not required. A simplified tabu in pseudocode is shown here:

1. tofix:=a random unsatisfied clause
2. FOR all variables in tofix
3. IF flipping variable breaks no clauses
4. THEN collect in best data structure
5. IF variable is not on tabu list
6. IF variable is best or equal-best seen so far
7. THEN collect in best data structure
8. choose randomly among variables in best

The variables collected in the best data structure are either the variables that break no clauses (if any such variables exist) or the variables with the best score when sorted by breaks. If the loop exits with nothing in best, another clause is tried. If all clauses produce the same result, the tabu list is ignored.

In the context of kernel clauses, we know that a variable can’t have more than one break, and also that there are two variables, so we can simplify the heuristic without losing any functionality (assuming kernel clauses are tagged and treated separately). The following code is specific to kernel clauses and it branches instead of looping so it is probably faster:

1. tofix:=a random unsatisfied clause with two variables, v1 and v2
2. IF v1 breaks no clauses
3. THEN IF v2 breaks no clauses
4. THEN choose between them randomly
5. ELSE choose v1
6. ELSE IF v2 breaks no clauses
7. THEN choose v2
8. ELSE IF v1 and v2 are on tabu list
9. THEN choose another clause
10. ELSE choose one of v1 or v2 according to tabu list

This still consults the tabu list, because tabu does somewhat better when the kernel tabu list is set to 5 than when it is 0 (from Figures 4.1, 4.2 and 4.3). For problem instance le450_25a, setting the kernel tabu list length to 5 and the other parameter optimally gives 14813 flips. Setting the length to 0 (equivalent to having no tabu list) and the other parameter optimally gives 17304 flips. In this case, having a tabu list reduces the number of flips required by 15%.
Removing the tabu list from the heuristic therefore would have to reduce the average flip time by 15% or more to be useful on this instance. If the tabu list were removed, the simplified code would look like this:

1. tofix:=a random unsatisfied clause with two variables, v1 and v2
2. IF v1 breaks no clauses
3. THEN IF v2 breaks no clauses
4. THEN choose between them randomly
5. ELSE choose v1
6. ELSE IF v2 breaks no clauses
7. THEN choose v2
8. ELSE choose between them randomly

This would be very fast since it no longer looks in the tabu list, and there are very few branches. Whether it would be fast enough to merit giving up the tabu list depends on its speed relative to the unmodified tabu. Similar modifications could possibly be performed for other problems; round robin clause types 2 and 3 listed in section 2.3.3 are also kernel clauses with two variables, so the pseudocode algorithm above would be appropriate for them. I have done no tests on round robin with tabu, so I don’t know how the setting of noise parameters for 2 and 3 affect the speed of the search. I would speculate that the effect would be similar to that of the kernel noise parameter of graph colouring, because the encodings are similar (in that they both have positive ALO clauses, no AMO clauses and negative kernel clauses and are both unary).

Although less promising than the new initializations in section 6.1.2, these modifications of tabu could be useful. The tabu list could also be removed when there are more than two literals in the kernel clause, which makes the idea applicable to all the kernel clauses in other unary SAT-encoded problems (including round robin).

### 6.3 A modified rnovelty

Section 5.2 considers (among other things) the graph colouring problem with rnovelty. A characteristic found was that with kernel noise set below 0.5, performance is terrible. This coincides with a discontinuity in the noise handling of rnovelty. In the case where the best variable (under the makes – breaks sort) is also the most recently flipped, the heuristic chooses probabilistically, with the probabilities shown in Figure 6.2 (n is the difference in makes – breaks between the best variable and the second-best variable). The only change in the heuristic between kernel noise 0.55 (which succeeded) and 0.5 (which failed) is that the probability of picking the second-best variable when n > 1 has changed from 0.05 to 0. This can be seen clearly in Figure 6.2. Hence, a logical improvement to try would be to limit that probability to the range 0.05-1. The effect of this modification, with instance le450_25a is shown in Figure 6.3. Figure 5.11 on page 46 is the equivalent chart for standard rnovelty.

As can be seen from the charts, the modification removes the characteristic completely, and the modified behaviour closely resembles that of novelty on the same problem instance (see Figure 5.1 on page 35). The difference in behaviour between novelty and modified rnovelty on this instance is that novelty has a steeper drop-off in performance as either of the noise parameters tend to zero, and also that modified rnovelty has a somewhat steeper slope towards the line where kernel noise=1.
Figure 6.2: Probability of the second-best variable being chosen in two cases

Figure 6.3: Instance le450_25a, solved with modified rnovelty
**R\textsubscript{novelty}** (modified or otherwise) shows an improvement of approximately 15\% over **novelty** on this instance. The best median falls on or very near the line where kernel noise = 1 for both the original and modified **novelty**.

The small change in **rnovelty** is clearly an improvement if only one noise parameter \((p)\) is used, where the optimum setting of \(p\) falls below 0.5 with the modified **rnovelty**, and hence with the standard **rnovelty**, \(p\) is within a range that performs very poorly. This is very likely for instance le450_15a, which performs very poorly with one noise parameter when using the original **novelty**.

Intuitively, the angular chart of Figure 6.2 is probably not optimal; the optimal is (in my opinion) more likely to be a smooth function of the noise parameter, since there is no justification for placing a discontinuity at any particular noise setting. The minor modification discussed above doesn’t remove the discontinuity, just alters the rules slightly to avoid one of its effects. Also, there is no justification for only considering the best and second-best variables, except that limiting the choice may improve the computational efficiency of the heuristic.

Chris Harris and Nadja Williams suggest a modified version of **rnovelty**, which, they claim, performs similarly on hard random 3-SAT with the optimal noise parameter, but outperforms **rnovelty** when the noise parameter is sub-optimal [11]. This eases the problem of choosing a noise parameter. The new heuristic is called **srnovelty**, and it improves on **rnovelty** by taking the goodness of choosing the second best variable into account. The goodness is ‘a scaled difference between the objective scores for the best and second best moves’ [11]. The intuition is that the second best variable should not be chosen if it leads to a very bad state. **Rnovelty** makes some attempt to do this, but is not as sophisticated. The success of **srnovelty** claimed by Harris and Williams is evidence that there is still significant scope for improvement within the Walksat architecture.

### 6.4 Other ways of dividing the clauses

This project has only looked at dividing the clauses according to the structure of encoded problems. Can multiple noise parameters be applied to other situations, for example where no division is suggested by the encoding? To do so would require some method of examining the clauses and finding a good way of dividing them into sets. There are a number of ways this might be done, some of which are described here.

#### 6.4.1 Examine the clause length

An instance of unforced random 2-SAT is solved fastest by **best** when the noise parameter is set to 1, and unforced random \(N\)-SAT has a declining optimal noise as \(N\) increases from 2 [7, chapter 4]. Mixing, for example, 2-SAT and 3-SAT over the same set of variables may reveal that the two types do have different optimal noise parameters despite the shared variables. This approach doesn’t extend to the highly structured problems considered in this project, since the clauses are highly non-random (e.g., ALO clauses are positive rather than a mixture of positive and negative). If successful, this approach may apply to SAT instances where the clauses contain a mixture of positive and negative literals, and have a mixture of lengths.

57
6.4.2 Look at the distribution of variables over clauses

Let $V$ be the set of variables found in the set of clauses $C$, where $C$ is a subset of a SAT instance. If $V$ are only seen within $C$, then $C$ is a sub-instance which is independent, and could possibly solve faster with a different noise setting to the rest of the problem instance. This is the situation described in section 2.2.2 on page 12. Of course, this is highly contrived, but the same principle could apply where variables in $V$ are restricted to $C$ but with some exceptions. Of course, developing some idea about how many exceptions can be allowed is difficult and would probably have to be done empirically. This approach could apply to encoded problems, where the sets of clauses can be determined during encoding, and the number of variables shared between sets is known during encoding.

6.4.3 Look at the characteristics of the clauses in the context of structure

Two SAT instances with the same set of variables can be morphed together to generate a third that has some of the characteristics of each. This is introduced by Gent, Hoos, Prosser and Walsh [13]. Their paper concentrates on mixing randomness and structure, with an example of the Manhattan grid: delivering parcels in Manhattan (where the street structure is extremely regular) is complicated by one-way streets, roadworks etc. These factors are considered random, and the grid is considered to be the structure.

Consider a real-world SAT instance where a set $A$ of clauses all fit some pattern (like the Manhattan grid) and a disjoint set $B$ appear to be random. If the structure is known beforehand it is possible to determine which clauses are in $A$ by comparing them to the structure, and all the others would be in $B$. It could be that the instance solves fastest when these two highly different sets of clauses are given different noise settings. Gent, Hoos, Prosser and Walsh experimented with graph colouring instances encoded as SAT, and observed that for Walksat the optimal noise setting is positively correlated with the amount of randomness in the graph [13]. This suggests that clauses in set $B$ may require more noise than those in $A$. Unfortunately they don’t report which heuristic they used, or the exact encoding, saying only that it is unary. This could apply to any SAT problem where the structure and exceptions are known.
Chapter 7

Discussion and Conclusions

7.1 Multiple noise parameters with encoding-based clause types

For the most part, this report has covered the idea of applying independent noise settings to the sets of clause produced by encoding problems into SAT. The objective has been to improve performance. Chapter 4 presented results for the best and tabu heuristics, and proved that best is totally unaffected by the setting of kernel noise parameters, for unary encodings. As a result, using multiple noise parameters is only as effective as using one, whenever a unary encoding is used (or another encoding whose clauses fit the conditions listed in section 4.1.1). The chapter also shows that tabu behaves in a similar way to best, and as a result it is unlikely that multiple noise parameters will provide a significant benefit for tabu.

Chapter 5 investigates the properties of the (r)novelty heuristics with multiple noise parameters. For novelty, two instances of graph colouring show a marked decrease in median flips when the kernel noise is set to 1 (and ALO noise is set optimally), and the third shows no significant difference. This is a useful observation, because it reduces the problem of finding optimal noise settings: the kernel noise is set to 1, and the ALO noise will probably be low, which reduces the range of possibilities. Of the three instances tested, all three had an optimal ALO noise setting of less than 0.5.

For rnovelty, the performance is awful for graph colouring when kernel noise is set below 0.5. This is investigated further in section 6.3, and a modified rnovelty is presented without this characteristic. Apart from this characteristic, rnovelty proved to be very similar in behaviour to novelty on the instances I tried. This is no surprise of course, rnovelty is a refinement of novelty.

For the round robin problem, (r)novelty shows no promise of improvement by using multiple noise parameters. An exhaustive search of the parameter space is infeasible in terms of CPU time, but from the experiments performed, it seems very likely that multiple noise parameters offer no improvement for the instance where n = 10, or in fact any instance where n ≥ 10.

7.2 Possible future directions

Chapter 6 suggested possible ways of extending this work. One interesting direction is specializing Walksat to take advantage of the different clause types produced by an encoding. Some ideas for improving the performance of best (given the property that the kernel noise
setting has no effect) were presented in section 6.1, one of which (novel initializations for unary SAT-encoded problems) applies to Walksat in general. Also, tabu can be tuned to unary SAT-encoded problems to improve its flip rate, and some ideas for doing this (applied to graph colouring) were presented.

Chapter 6 also discusses using multiple noise parameters, but dividing the clauses in another way not related to an encoding. Three methods are presented, all of which are (in my opinion) feasible, and it is hard to say which one is the most promising without some empirical evaluation. Continuing with multiple noise parameters is one possibility for future work.

The results discussed in section 7.1 do not directly apply to local search procedures other than Walksat. Multiple noise parameters for different clause types could not be applied to them, because they pick variables directly. However, the idea of using other features of a set of instances to optimize aspects of the search (such as providing better initialization) applies to any local search procedure.

Schuurmans and Southey [8] attempted to identify the characteristics of a local search algorithm when it is working well. They defined three, depth, mobility and coverage. Perhaps the best idea for the long term is rewriting the local search procedure to manage the tradeoff between these three (and possibly other characteristics that have not yet been identified), without needing a noise parameter at all. This is the aim of the SDF algorithm [8], which takes two parameters, neither of which are for setting noise (even so, the ideal algorithm would not have parameters).

I speculate that one of the most promising avenues for future research is to specialize Walksat to sets of instances, not using just the idea of multiple noise parameters, but other insights specific to the set of instances (such as the novel initializations mentioned above), to eliminate wasted work and optimize the behaviour of Walksat.
Bibliography


