Improving UnitMarch

M. Jepson* M.A. Mol†

June, 2007

Abstract

UnitMarch is a solver for the satisfiability problem, based on UnitWalk. The only real difference between the two is the fact that UnitMarch uses a 32 bit approach, which makes it effectively try 32 different possible solutions at once. Both solvers use unit clause elimination and random flipping to work towards a solution, but neither versions use any heuristics.

This research tries to improve UnitMarch by adding some heuristics to the algorithm. Different approaches for improving the algorithm are addressed and tested. From the results of these tests the best approaches are selected to construct an improved version of the UnitMarch algorithm.

keywords: *satisfiability problem, SAT, UnitWalk, UnitMarch, SAT solver*

1 Introduction

UnitMarch[1] is basically a 32 bit version of UnitWalk[2], without any further improvements. As such, the main goal of this research is to add heuristics to the algorithm, in the hope that it will speed up the process.

The next two subsectionts will first describe the algorithm of UnitWalk and the difference between UnitWalk and Unitmarch.

Section 2 lists the heuristic opportunities for improving UnitMarch's algorithm, the results of which are listed in section 3.

1.1 UnitWalk

A well known local search algorithm, designed and implemented by Hirsch and Kojevnikov[2] is UnitWalk. Their approach was to design an incomplete algorithm based on unit clause elimination.

1.1.1 The UnitWalk algorithm

The algorithm starts with making a random truth assignment A. The next step is to make a random permutation $\pi$of all the variables in the formula.

First the algorithm looks for any unit clauses (clauses with only one variable) in the formula. If there are such clauses, they are satisfied by setting the variable to the correct value and removed from the formula. The value of the variable is also stored in A. If there are no such clauses, or all unit clauses have been processed, the next variable in π is set to **true** or **false** according to its value in A and all clauses that are satisfied by this variable are removed from the formula. This variable is also removed from all clauses which are not satisfied by this variable.

When this is done, the algorithm begins with checking for unit clauses again.

When the end of the permutation is reached, the algorithm checks if the formula has any clauses left, if not, it is satisfied by A and A is returned as a solution to the problem. If no changes to A are made during the period, one of A's values is flipped at random.

*TU Delft, Netherlands, student#: 1058916. E-mail: mjepson@gmail.com

†TU Delft, Netherlands, student#: 1051881. E-mail: martijn.mol@gmail.com

Algorithm 1 Pseudocode of the UnitWalk algorithm

```
1: for t := 1 to MAX_TRIES do
2:   A := random truth assignment for n variables;
3:   for p := 1 to MAX_PERIODS do
4:     π := random permutation of 1..numberOfVariables;
5:     G := copy of the original formula
6:     f := 0;
7:     for i := 1 to numberOfVariables do
8:       while G contains a unit clause, repeat do
9:         Pick a unit clause {x_j} or {¬x_j} from G at random;
10:        if this clause is not satisfied by A and G does not contain the opposite unit clause
           then
11:          flip A[j];
12:          set f := 1;
13:        G := G[x_j] A[j];
14:      if variable x_π[i] still appears in G then
15:        G := G[x_π[i] ← A[π[i]]];
16:    if G contains no clauses (i.e., G ≡ True) then
17:      output A
18:      exit;
19:    if f = 0 then
20:      choose a variable at random;
21:      flip its truth value;
22: Output "Not found".
```

After this, the formula is restored to its original form and the whole process is restarted, with A kept as it is. When *MAX_PERIODS* is reached, the algorithm starts with a new *try*, until *MAX_TRIES* is reached.

1.2 UnitMarch

M. Heule and H. van Maaren[1] have developed UnitMarch, an algorithm based on UnitWalk. Unlike UnitWalk, UnitMarch uses multi bit assignments. Each truth value in such an assignment represents 32 different possible truth values for a variable. Heule and van Maaren describe in their paper algorithms for flipping and unit propagating multi bit assignments. They restructured UnitWalk to support multi bit assignments by using 32 bit integer numbers in stead of single bit booleans to represent truth values. The multi bit truth assigment basically represents 32 different single bit assignments or simply 32 different assignments. The main idea is that these 32 different assignments (and possible solutions) are processed in parallel, making efficient use of the 32 bit nature of most computer processors. Because of this approach, UnitMarch performs better on most formulas than UnitWalk does. Despite the overhead created to support 32 bit truth values, the results of the 1 bit version are comparable to those of UnitWalk. As computer processors evolve and start working with higher numbers of bits per instruction, this approach can easily be expanded to 64 or even more bits, to obtain an even higher efficiency.

Besides some new internal structures and methods for supporting the 32 bit assignments, the main idea of the algorithm remains the same to that of UnitWalk, and for simplicity all changes to the algorithm will be explained using the algorithms of the latter.

1.2.1 Removing duplicate assignments

A first possible improvement that came to mind was removing any duplicate assignments. Due to the unit propagations in each period, some assignments might become identical. Identical assignments will always remain identical, as they are altered in exactly the same way.

When two or more assignments are identical, performance decreases, as it is like running two identical instances of UnitWalk with the same initial assignment.

By removing duplicate assignments and replacing them with new randomly chosen ones, performance can be improved. Algorithm 1 was altered by placing the code in algorithm 2 after line 21.

Algorithm 2 Removing duplicate assignments

```
1: for all different assignments do
2:   for all variables do
3:     Check which assignments are the same for the current variable and have been for all other
       variables;
4: if duplicates exist then
5:   replace all but one copy of it with random assignments;
```

Removing duplicate assignments is an alteration that adds an $O(b*v)$ overhead to the algorithm, with b being the number of bits in the multi bit assignment and v the number of variables in the formula.

As this is not a heuristic and as its positive influence on execution time is obvious, this alteration is now a standard part of the UnitMarch algorithm and therefore listed in the introduction part of this paper.

2 Heuristic opportunities

This section describes some theories of how to make the UnitMarch algorithm even faster. The results of testing these theories are listed in section 3.

2.1 Propagation order

The original algorithm of both UnitWalk and UnitMarch started each period with a random ordering π. This permutation is the order in which all variables are processed during the period. There might be some room for improvement here, and one possibility is to make the permutation less random. One way of doing that is making the permutation depend on how often a variable appears in unsatisfied clauses.

This was achieved by giving the variables a so-called *appearance* score, which represented how many times it appeared in an unsatisfied clause. Two different approaches were used, one by raising the *appearance* with a certain *weight* each time it was encountered in unsatisfied clauses, the other by multiplying it with a certain *factor*. For creating the permutation π, these *appearances* were multiplied with a random number, and then the whole list of variables was sorted descending according to that outcome.

A possible gain in speed was expected here, since this approach forces the algorithm to focus more on the unsatisfied clauses, but still leaves room for the random aspect of the UnitWalk idea. For testing this theory, the source code was altered by replacing line 4 of algorithm 1 with the code in algorithm 3.

As seen in the code, this alteration adds an $O(b * v)$ overhead to the algorithm, with b being the number of bits and v being the number of variables in the input formula.

2.2 Limited periods per try

Each *try* starts with a random truth assignment A which is continuously altered during the *periods* in the *try*. If *MAX_PERIODS* is reached, the algorithm starts a new *try*. For problems with a lot of different possible solutions, the algorithm might be faster when it performs more *tries* and less *periods* per *try*.

Algorithm 3 The new propagation order

```
1: for all the clauses in the formula do
2:    if clause is unsatisfied then
3:       for all the variables in the clause do
4:          appearance for this variable += weight; (if weight is declared)
5:          appearance for this variable *= factor; (if factor is declared)
6: perm := empty array;
7: for all the variables in the formula do
8:    add appearance*random_factor to perm;
9: π := perm sorted descending;
```

In its current form, the algorithm runs with $MAX\_PERIODS = \infty$. One obvious alteration is of course giving this and *MAX_TRIES* different values.

2.2.1 Increasing number of periods per try

Because both *MAX_TRIES* and *MAX_PERIODS* are 'magic numbers' in the above alteration, a possible increase in performance might be obtained here. So the following idea was produced. The algorithm starts out with a *try* with very little *periods* in it, and the *tries* get longer and longer each time a new *try* begins. The new algorithm was produced by replacing the first two lines of algorithm 1 with the code in algorithm 4.

Algorithm 4 Increasing periods per try

```
1: for t := 1 to MAX_TRIES do
2:    A := random truth assignment for n variables;
3:    MAX_PERIODS := 10 * 1.5^t;
```

This change produces no overhead at all, as it is just a different approach to assigning numbers to *MAX_TRIES* and *MAX_PERIODS*.

2.3 Removing bad assignments

UnitMarch in its 32 bit form has 32 presumably different assignments in memory. These different assignments have no knowledge about each other and all work towards a possible solution to the problem separately. One possible improvement might be to keep the best assignments, and re-initiate the others with random assignments. To test this, the algorithm was again altered so that it would perform this action every *AMOUNT_OF_PERIODS_BEFORE_REMOVE_WORST* periods. The alteration consisted of replacing lines 19-21 in algorithm 1 with the code in algorithm 5.

Algorithm 5 Removing the worst assignments

```
1: if p modulo AMOUNT_OF_PERIODS_BEFORE_REMOVE_WORST = 0 then
2:    for all clauses in the formula do
3:       for all 32 assignments do
4:          if the assignment does not satisfy the clause then
5:             raise its counter;
6:    Remove the AMOUNT_OF_ASSIGNMENTS_TO_REMOVE worst assignments and replace
      them with random assignments; (the higher its counter, the worse the assignment does)
```

This addition to the algorithm adds again an $O(b * c * v)$ overhead, with b being the number of bits and c being the number of clauses and v being the number of variables in the input formula.

2.3.1 Changing kept assignments

To prevent certain assignments from getting trapped in suboptimal solutions, they can be altered with a form of random noise. The algorithm was altered so that every time the worst assignments were dropped, the other assignments were altered by flipping each truth value in it with a probability c. The code in algorithm 6 was added to the end of 5, within the **if**-statement opened in line 1.

Algorithm 6 Changing kept assignments

```
1: for all variables do
2:     Flip its truth value with probability c;
```

This alteration only adds an $O(v)$ overhead to the algorithm, with v being the number of variables in the input formula.

3 Results

This section follows the same order as section 2 and lists the gained or lost performance for each alteration. All parts of the appendix referred to have the first table listing the results from running the standard 32 bit version of UnitMarch on the same machine, for comparison of the results.

3.1 Removing duplicate assignments

As this alteration can only have a positive influence on the performance of UnitMarch, no actual performance comparison was made between the versions with and without this alteration. This slight change is now a standard part of the UnitMarch algorithm, and as such is incorporated in all results listed in the appendices.

3.2 UnitMarch VS. UnitWalk

As seen in appendix B.1, the 1 bit version of UnitMarch's performance is at least comparable to that of UnitWalk. So the alterations in the internal algorithms to support the multi-bit nature of UnitMarch do not have a negative impact on performance.

The 32 bit version of UnitMarch performs significantly better than UnitWalk, making this the better of the two.

3.3 New propagation order

Appendix B.2 lists all the results from testing the new propagation.

Section B.2.1 lists the results of using different **weights** for calculating the *appearance* score of each variable used for creating the propagation permutation π as seen in algorithm 3.

As seen in the results, this approach gives good results for the aim-200-2_0 series, and especially for the logistics series, but it has large negative influences on the par16 series. Overall results are not convincing, as improvements for some series are cancelled out by worse performances on others.

The results of the using a **factor** for calculating the *appearance* scores are listed in appendix B.2.2.

This approach gives an improved or comparable results for all sets, except for the par16 sets, qg5-11 and qg7-13. The best improvements were acquired with the factor set to two or three, as these give the most stable results (slight improvement overall, rather than a big improvement in some sets, and a big disimprovement in others).

3.4 Limited periods per try

The results from this alteration are listed in appendix B.3 Overall, limiting the periods performed worse than not limiting it. When *MAX_PERIODS* is very small, performance suffers badly.

3.4.1 Increasing number of periods per try

This change in the algorithm gave surprisingly good results as can be seen in appendix B.3.1. Overall this version performed better than the standard UnitMarch, except for the bw_large.c, par16-2, qg1-08 and uf250-071 sets.

With the exception of qg1-08, all these disimprovements were only minor, while the performance showed particular improvements on aim-200-3_4-yes1-2 and qg2-08.

3.5 Removing bad assignments

See appendix B.4 for the results. Overall the algorithm did not perform better than without this alteration. Especially the [28,10] version performs bad. The version with [28,100] performed slightly better on some sets, although not convincingly.

3.5.1 Changing kept assignments

As can be seen in the results in appendix B.4.1, damaging the remaining assignments did not improve the algorithm on the overall. As with the limiting periods per try and removing bad assignments, setting the amount of periods very low gives very poor performance. These results were not much of a surprise as they are quite similar to limiting the amount of periods per try.

4 Conclusion

With the results addressed in section 3, some positive alterations to the UnitMarch algorithm can be selected. Adjusting the propagation order did have some positive effects of the algorithm. Using a factor in the the algorithm gave better performance than using a weight. Also changing the number of periods per try from a static number to a variable number gave somewhat positive results.Removing bad assignments and changing intermediate assignments did not have a positive effect on the performance of the algorithm.

The final algorithm is based on UnitMarch's 32 bits version with removing duplicate assignments. It incorporates the adjusted propagation algorithm and the variable number of periods per try, as these were the most convincing improvements. Results from this improved algorithm are shown in appendix C. From the results can be concluded that the final version is an improvement of the UnitMarch algorithm.

5 Discussion and future work

Ten runs on each test set might be too small for the algorithm to come up with good and valid results. As such, results might differ when the algorithm is allowed to do more runs on each test set. These tests might even come up with such different results such that our choices for the final improved version become invalid or suboptimal. So a first step in future work should be to perform these tests.

Furthermore, in this research, more 'magic' numbers have been used, especially in the results described in section 3.5. These tests should also be expanded to other possibilities, like [24,250] for instance.

References

[1] M. Heule H. van Maaren. From idempotent generalized boolean assignments to multi-bit search.

[2] E.A. Hirsch, A. Kojevnikov. Unitwalk: A new sat solver that uses local search guided by unit clause elimination. *Annals of Mathematics and Artificial Intelligence*, 43(1):91–111, 2005.

A Hardware

Two different computers were used for testing. The specifications of both are listed below:

- CPU: AMD Athlon XP 2100+ @ 1733MHz
- Memory: 512 MB @ 266 MHz
- OS: Mandriva Linux 2007 (Free edition), based on the 2.6 kernel

Table 1: specifications of the machine known as '2100'

- CPU: AMD Athlon XP 2800+ @ 2250MHz
- Memory: 1024 MB @ 333 MHz
- OS: Mandriva Linux 2007 (Powerpack), based on the 2.6 kernel

Table 2: specifications of the machine known as '2800'

B Results

These results are all based on running the algorithm ten times on the set. Each run was initiated with a different seed. The hardware used for this test is describe in appendix A.

B.1 UnitWalk & UnitMarch

The test results below have been computed using the '2800' computer described in appendix A and the algorithms described in section 1.1 and 1.2.

Test Set	UnitWalk 1.03		UnitMarch_1_bits		UnitMarch_32_bits	
	Periods	Time	Periods	Time	Periods	Time
4blocks[1]	312671.6	>467.19	22692.0	$115.05^{(161.69)}$	884.3	$13.63^{(8.70)}$
aim-200-2_0-yes1-1	125698.4	$6.36^{(4.70)}$	29144.4	$2.10^{(1.95)}$	1509.9	$0.49^{(0.57)}$
aim-200-2_0-yes1-2	1870656.8	$96.59^{(90.39)}$	1022260.0	$72.90^{(54.98)}$	39442.4	$13.10^{(14.12)}$
aim-200-3_4-yes1-1	103152.5	$8.81^{(7.46)}$	53303.5	$4.38^{(3.76)}$	2321.3	$2.13^{(1.81)}$
aim-200-3_4-yes1-2	104150.9	$8.82^{(8.41)}$	293108.0	$26.13^{(24.72)}$	7016.4	$6.42^{(6.51)}$
aim-200-3_4-yes1-4	191145.3	$16.50^{(12.06)}$	97451.1	$8.01^{(3.96)}$	3848.1	$3.55^{(2.43)}$
ais12	29752.4	$4.03^{(3.13)}$	24043.8	$7.24^{(6.53)}$	635.6	$1.10^{(1.52)}$
bmc-ibm-5	1464.9	$16.50^{(11.73)}$	8639.6	$82.97^{(62.00)}$	499.5	$9.43^{(8.13)}$
bw_large.c	12950.3	$56.80^{(68.44)}$	20685.2	$99.93^{(53.86)}$	549.7	$11.76^{(8.82)}$
logistics.a[1]	1284206.1	>463.11	522613.0	>456.12	26281.6	$79.87^{(59.66)}$
logistics.c	101567.3	$52.82^{(42.17)}$	191006.0	$301.65^{(302.37)}$	5632.7	$27.30^{(25.90)}$
par16-1	9761.4	$4.25^{(4.69)}$	8699.5	$3.16^{(2.36)}$	403.3	$0.32^{(0.44)}$
par16-2	21096.3	$9.24^{(8.53)}$	13310.7	$4.82^{(2.56)}$	411.9	$0.33^{(0.25)}$
par16-3	25039.3	$10.95^{(9.71)}$	16884.9	$6.06^{(4.45)}$	669.3	$0.53^{(0.68)}$
par16-4	11667.8	$5.06^{(5.72)}$	14148.5	$4.65^{(6.13)}$	419.3	$0.33^{(0.33)}$
qg1-08[1]	120382.5	>641.57	52052.1	>729.02	3772.7	$198.68^{(180.79)}$
qg2-08[1]	142471.6	>762.84	60434.4	>844.79	23735.6	$1193.09^{(1126.30)}$
qg5-11	979.2	$3.43^{(3.46)}$	737.8	$4.97^{(4.73)}$	16.4	$0.57^{(0.75)}$
qg7-13[1]	93792.5	>532.00	39543.4	>433.09	2855.1	$165.08^{(131.52)}$
uf250-054	350077.5	$42.16^{(42.71)}$	592624.0	$79.67^{(106.37)}$	7932.9	$10.26^{(8.71)}$
uf250-062	67930.3	$8.14^{(6.13)}$	41962.4	$5.66^{(3.34)}$	4197.7	$5.41^{(4.97)}$
uf250-071	154557.6	$18.70^{(22.27)}$	223382.0	$28.82^{(14.42)}$	3487.9	$4.49^{(3.01)}$
uf250-072	79596.6	$9.70^{(7.17)}$	90263.4	$12.20^{(12.84)}$	5936.4	$7.82^{(8.85)}$
uf250-093	122762.8	$14.52^{(22.88)}$	140526.0	$18.41^{(17.62)}$	2825.1	$3.63^{(2.64)}$

[1]Some results for these sets are invalid (indicated with >), as they hit their 900 second maximum running time.

B.2 Propagation order

The test results below have been computed using the '2100' computer described in appendix A and the algorithm described in section 2.1.

B.2.1 Weight

Test Set	UnitMarch_32_bits		UnitMarch_32_bits (W=5)[2]		UnitMarch_32_bits (W=10)[2]		UnitMarch_32_bits (W=25)[2]	
	Periods	Time	Periods	Time	Periods	Time	Periods	Time
4blocks	884.3	$21.62^{(13.32)}$	569.4	$20.87^{(19.95)}$	637.4	$26.75^{(20.46)}$	1033.2	$35.31^{(34.64)}$
aim-200-2_0-yes1-1	1509.9	$0.65^{(0.74)}$	275.5	$0.15^{(0.12)}$	402.5	$0.29^{(0.18)}$	335.2	$0.18^{(0.11)}$
aim-200-2_0-yes1-2	39442.4	$17.28^{(17.67)}$	8266.4	$4.25^{(4.01)}$	7369.7	$4.81^{(3.55)}$	7270.1	$3.65^{(2.54)}$
aim-200-3_4-yes1-1	2321.3	$2.79^{(2.26)}$	1745.8	$2.25^{(1.39)}$	2797.3	$4.59^{(3.91)}$	938.3	$1.28^{(1.01)}$
aim-200-3_4-yes1-2	7016.4	$8.41^{(8.07)}$	10518.5	$13.43^{(9.93)}$	7142.4	$11.51^{(13.37)}$	6614.7	$8.33^{(5.65)}$
aim-200-3_4-yes1-4	3848.1	$4.63^{(3.00)}$	3724.0	$4.77^{(2.95)}$	5620.1	$9.14^{(7.52)}$	3680.9	$4.99^{(5.53)}$
ais12	635.6	$1.75^{(2.29)}$	250.0	$0.84^{(0.77)}$	268.5	$1.26^{(1.38)}$	231.2	$0.83^{(0.85)}$
bmc-ibm-5	499.5	$16.08^{(13.13)}$	360.1	$15.11^{(5.78)}$	217.9	$11.71^{(9.08)}$	471.3	$21.51^{(10.95)}$
bw_large.c	549.7	$21.32^{(15.19)}$	241.8	$11.13^{(8.34)}$	472.0	$21.74^{(16.79)}$	328.8	$15.55^{(12.42)}$
logistics.a	26281.6	$128.15^{(103.02)}$	3236.6	$17.37^{(10.08)}$	1284.5	$7.39^{(10.45)}$	1706.1	$9.99^{(6.28)}$
logistics.c	5632.7	$44.38^{(42.32)}$	182.6	$1.68^{(0.95)}$	155.6	$1.86^{(2.06)}$	251.2	$2.45^{(1.42)}$
par16-1	403.3	$0.96^{(1.49)}$	307.4	$0.66^{(0.55)}$	270.7	$0.80^{(0.80)}$	281.4	$0.68^{(0.60)}$
par16-2	411.9	$0.72^{(0.54)}$	1301.1	$2.86^{(3.64)}$	1776.8	$5.58^{(3.75)}$	1779	$4.21^{(5.24)}$
par16-3	669.3	$1.24^{(1.34)}$	972.8	$2.10^{(2.06)}$	1147.6	$3.17^{(2.68)}$	967.4	$2.27^{(2.92)}$
par16-4	419.3	$1.17^{(1.23)}$	769.4	$1.67^{(1.69)}$	933.5	$2.81^{(2.22)}$	1559.1	$3.65^{(5.14)}$
qg1-08	3772.7	$359.21^{(307.34)}$	3507.6	$341.81^{(336.34)}$	2797.1	$287.88^{(229.68)}$	2914.7	$293.11^{(360.09)}$
qg2-08	23735.6	$2157.08^{(2036.33)}$	50044.9	$4872.97^{(3504.81)}$	30588.9	$3059.29^{(3528.59)}$	29600.7	$3005.53^{(2626.09)}$
qg5-11	16.4	$0.99^{(1.25)}$	151.7	$9.61^{(12.79)}$	133.3	$9.12^{(9.09)}$	68.5	$4.86^{(3.44)}$
qg7-13	2855.1	$269.58^{(194.66)}$	4181.1	$428.68^{(338.83)}$	4342.7	$498.45^{(385.64)}$	2607.8	$326.44^{(230.47)}$
uf250-054	7932.9	$13.48^{(10.84)}$	4908.9	$9.46^{(6.69)}$	5255.2	$11.01^{(8.64)}$	4564.2	$11.9^{(11.52)}$
uf250-062	4197.7	$7.14^{(6.19)}$	2899.8	$5.63^{(6.83)}$	2905.5	$6.23^{(5.23)}$	2440.7	$5.49^{(4.92)}$
uf250-071	3487.9	$5.89^{(3.76)}$	4809.9	$9.36^{(9.14)}$	4323.6	$9.05^{(6.71)}$	2641.4	$6.46^{(5.57)}$
uf250-072	5936.4	$10.32^{(11.02)}$	4249.1	$8.38^{(5.17)}$	1411.9	$3.04^{(1.88)}$	1600	$3.73^{(2.91)}$
uf250-093	2825.1	$4.81^{(3.32)}$	2725.3	$5.27^{(4.71)}$	4204.4	$8.81^{(6.3)}$	1816.3	$6.78^{(5.40)}$

[2] In these tests the 'New permutation', based on the appearances of variables in unsat clauses is used. (W) means APPEARANCE_WEIGHT was defined, (F) that APPEARANCE_FACTOR was defined.

B.2.2 Factor

Test Set	UnitMarch_32_bits (F=2)[3]		UnitMarch_32_bits (F=3)[3]		UnitMarch_32_bits (F=4)[3]		UnitMarch_32_bits (F=5)[4]		UnitMarch_32_bits (F=10)[4]	
	Periods	Time	Periods	Time	Periods	Time	Periods	Time	Periods	Time
4blocks	835.4	$25.93^{(19.20)}$	849.0	$52.81^{(36.20)}$	588.6	$21.97^{(18.07)}$	1005.0	$48.95^{(29.13)}$	874.9	$28.68^{(26.32)}$
aim-200-2_0-yes1-1	360.5	$0.17^{(0.12)}$	213.5	$0.17^{(0.15)}$	228.3	$0.11^{(0.09)}$	201.6	$0.12^{(0.10)}$	84.4	$0.04^{(0.04)}$
aim-200-2_0-yes1-2	12382.8	$6.02^{(6.91)}$	4631.6	$4.03^{(3.37)}$	3347.2	$1.76^{(1.38)}$	4696.9	$3.36^{(3.15)}$	2927.2	$1.40^{(1.32)}$
aim-200-3_4-yes1-1	1446.6	$1.77^{(2.07)}$	986.4	$2.31^{(2.73)}$	1357.2	$1.81^{(2.12)}$	1138.8	$1.65^{(1.70)}$	1101.8	$1.35^{(1.56)}$
aim-200-3_4-yes1-2	3578.9	$4.41^{(3.68)}$	4388.2	$10.51^{(12.04)}$	3764.4	$5.06^{(3.00)}$	5880.7	$10.07^{(15.35)}$	4324.7	$5.27^{(3.69)}$
aim-200-3_4-yes1-4	5654.6	$6.94^{(6.59)}$	2056.5	$5.15^{(4.07)}$	4352.4	$6.00^{(5.58)}$	3483.6	$5.90^{(5.50)}$	2366.1	$2.90^{(2.68)}$
ais12	311.7	$0.98^{(0.81)}$	451.1	$2.60^{(2.35)}$	304.0	$1.02^{(0.82)}$	164.5	$0.70^{(0.79)}$	430.6	$1.35^{(0.67)}$
bmc-ibm-5	377.8	$15.46^{(6.70)}$	502.2	$39.37^{(23.50)}$	446.3	$22.72^{(24.62)}$	499.2	$25.53^{(20.71)}$	920.0	$37.10^{(27.70)}$
bw_large.c	209.2	$8.21^{(7.55)}$	205.0	$10.26^{(7.88)}$	383.8	$16.66^{(24.07)}$	245.2	$10.82^{(6.52)}$	378.7	$15.42^{(10.22)}$
logistics.a	1890.6	$9.63^{(8.17)}$	1003.7	$5.43^{(3.74)}$	576.3	$3.94^{(3.90)}$	605.3	$3.50^{(2.03)}$	1288.4	$6.76^{(6.42)}$
logistics.c	161.6	$1.38^{(1.19)}$	186.7	$1.72^{(1.58)}$	276.2	$2.53^{(2.16)}$	112.0	$1.09^{(0.54)}$	175.2	$1.53^{(1.10)}$
par16-1	307.0	$0.61^{(0.62)}$	429.5	$0.93^{(0.64)}$	195.4	$0.42^{(0.23)}$	251.4	$0.59^{(0.50)}$	180.1	$0.38^{(0.27)}$
par16-2	1969.1	$4.10^{(3.81)}$	800.1	$1.82^{(1.28)}$	1760.5	$3.90^{(3.08)}$	1348.4	$3.26^{(2.30)}$	1421.0	$3.10^{(3.11)}$
par16-3	1049.5	$2.09^{(1.72)}$	1227.0	$2.60^{(2.31)}$	1037.8	$2.21^{(2.09)}$	902.2	$2.02^{(2.02)}$	930.4	$1.91^{(1.35)}$
par16-4	706.2	$1.45^{(1.01)}$	784.7	$1.72^{(0.69)}$	1174.6	$2.56^{(2.64)}$	744.7	$1.75^{(0.79)}$	826.3	$1.76^{(1.23)}$
qg1-08	1819.8	$161.54^{(131.75)}$	2358.1	$234.78^{(156.54)}$	6113.6	$585.68^{(465.28)}$	2357.6	$238.90^{(256.21)}$	1628.8	$149.29^{(122.41)}$
qg2-08	12883.2	$1167.54^{(542.66)}$	9029.4	$1024.00^{(625.08)}$	26280.0	$2424.88^{(1564.69)}$	18668.8	$1771.00^{(1412.18)}$	27135.8	$2532.08^{(1780.08)}$
qg5-11	34.5	$2.11^{(1.94)}$	42.0	$2.73^{(2.25)}$	16.7	$0.96^{(0.68)}$	62.2	$3.69^{(3.04)}$	41.0	$2.46^{(2.17)}$
qg7-13	2485.0	$268.04^{(190.39)}$	4063.6	$427.63^{(426.61)}$	4102.1	$391.78^{(503.51)}$	4801.2	$473.51^{(358.77)}$	6066.9	$598.59^{(410.37)}$
uf250-054	2642.6	$4.88^{(3.18)}$	2117.0	$4.09^{(4.14)}$	3433.4	$6.12^{(4.98)}$	3099.7	$5.62^{(7.01)}$	4775.2	$8.58^{(7.67)}$
uf250-062	2686.1	$4.90^{(4.44)}$	1790.0	$3.51^{(2.75)}$	1432.1	$2.55^{(2.36)}$	2424.0	$4.40^{(6.55)}$	1986.1	$3.58^{(4.38)}$
uf250-071	2582.7	$4.68^{(4.32)}$	1420.4	$2.73^{(2.35)}$	2073.8	$3.65^{(2.98)}$	2818.7	$5.08^{(4.78)}$	1584.3	$2.87^{(1.86)}$
uf250-072	1953.6	$3.68^{(2.14)}$	1232.3	$2.43^{(1.17)}$	2106.3	$3.80^{(1.99)}$	3811.0	$7.07^{(6.71)}$	2492.4	$4.67^{(3.82)}$
uf250-093	2950.4	$5.39^{(5.77)}$	1575.9	$3.05^{(2.53)}$	3707.5	$6.55^{(9.37)}$	2060.0	$3.74^{(3.87)}$	2309.1	$4.16^{(3.42)}$

[4]In these tests the 'New permutation', based on the appearances of variables in unsat clauses is used. (W) means APPEARANCE_WEIGHT was defined, (F) that APPEARANCE_FACTOR was defined.

B.3 Limited periods per try

The test results below have been computed using the '2800' computer described in appendix A and the algorithm described in section 2.2.

Test Set	UnitMarch_32_bits		UnitMarch_32_bits [1000][5]		UnitMarch_32_bits [100][5]		UnitMarch_32_bits [10][5]	
	Periods	Time	Periods[6]	Time	Periods[7]	Time	Periods	Time
4blocks[8]	884.3	$13.63^{(8.70)}$	525.0	$15.02^{(8.35)}$	76.4	$41.71^{(38.27)}$	60921.4	>870.08
aim-200-2_0-yes1-1	1509.9	$0.49^{(0.57)}$	380.0	$0.41^{(0.27)}$	77.3	$0.25^{(0.15)}$	6577.4	$2.34^{(2.02)}$
aim-200-2_0-yes1-2	39442.4	$13.10^{(14.12)}$	557.2	$11.77^{(8.55)}$	61.4	$5.99^{(3.30)}$	322726.0	$139.14^{(97.92)}$
aim-200-3_4-yes1-1	2321.3	$2.13^{(1.81)}$	368.8	$1.91^{(2.10)}$	50.6	$1.25^{(0.57)}$	1417.9	$1.65^{(1.16)}$
aim-200-3_4-yes1-2	7016.4	$6.42^{(6.51)}$	490.9	$3.11^{(2.34)}$	54.0	$4.55^{(4.33)}$	5149.2	$5.13^{(3.15)}$
aim-200-3_4-yes1-4	3848.1	$3.55^{(2.43)}$	438.5	$4.23^{(2.60)}$	56.6	$6.81^{(5.99)}$	8901.1	$8.25^{(6.26)}$
ais12	635.6	$1.10^{(1.52)}$	402.8	$1.06^{(1.07)}$	59.8	$0.61^{(0.56)}$	1628.8	$2.75^{(2.33)}$
bmc-ibm-5	499.5	$9.43^{(8.13)}$	351.3	$13.83^{(13.20)}$	55.5	$12.25^{(10.41)}$	10831.3	$221.94^{(210.28)}$
bw_large.c	549.7	$11.76^{(8.82)}$	485.6	$16.80^{(13.67)}$	49.6	$12.64^{(13.14)}$	574.7	$12.38^{(5.73)}$
logistics.a[8]	26281.6	$79.87^{(59.66)}$	547.1	$74.34^{(83.41)}$	79.7	$104.39^{(104.11)}$	260774.0	>899.94
logistics.c[8]	5632.7	$27.30^{(25.90)}$	515.4	$29.79^{(26.90)}$	79.9	$103.84^{(64.09)}$	161283.0	>899.95
par16-1	403.3	$0.32^{(0.44)}$	246.1	$0.28^{(0.31)}$	51.9	$0.23^{(0.16)}$	469.1	$0.43^{(0.37)}$
par16-2	411.9	$0.33^{(0.25)}$	411.9	$0.34^{(0.24)}$	54.4	$0.37^{(0.29)}$	798.6	$0.72^{(0.55)}$
par16-3	669.3	$0.53^{(0.68)}$	293.3	$0.40^{(0.34)}$	35.0	$0.33^{(0.30)}$	910.8	$0.79^{(1.05)}$
par16-4	419.3	$0.33^{(0.33)}$	230.9	$0.37^{(0.36)}$	67.4	$0.45^{(0.50)}$	345.6	$0.31^{(0.27)}$
qg1-08[8]	3772.7	$198.68^{(180.79)}$	471.1	$282.53^{(219.47)}$	48.9	$143.67^{(179.88)}$	5543	>315.19
qg2-08[8]	123735.6	$1193.09^{(1126.30)}$	441.8	>760.12	47.8	>760.01	9908.5	$578.82^{(334.94)}$
qg5-11	16.4	$0.57^{(0.75)}$	16.4	$0.59^{(0.74)}$	16.4	$0.58^{(0.72)}$	20.7	$0.77^{(0.71)}$
qg7-13	2855.1	$165.08^{(131.52)}$	381.3	$207.68^{(170.06)}$	48.4	$165.25^{(112.88)}$	2456.8	$152.33^{(121.05)}$
uf250-054	7932.9	$10.26^{(8.71)}$	574.7	$19.90^{(25.60)}$	68.6	$21.13^{(12.27)}$	27779.1	$38.85^{(30.37)}$
uf250-062	4197.7	$5.41^{(4.97)}$	571.0	$4.14^{(2.92)}$	65.6	$4.06^{(4.51)}$	3742.9	$5.24^{(2.74)}$
uf250-071	3487.9	$4.49^{(3.01)}$	439.3	$4.16^{(2.43)}$	41.9	$10.76^{(10.81)}$	15114.7	$21.06^{(22.26)}$
uf250-072	5936.4	$7.82^{(8.85)}$	563.1	$4.17^{(3.68)}$	60.6	$8.43^{(6.60)}$	13534.9	$19.31^{(16.24)}$
uf250-093	2825.1	$3.63^{(2.64)}$	471.1	$6.78^{(7.12)}$	45.8	$3.52^{(4.05)}$	13148.4	$18.37^{(12.59)}$

[5] [x] means that maximum periods per try was set to x.

[6] The periods listed are not the absolute number of periods, but due to testing problems mod(1000)

[7] The periods listed are not the absolute number of periods, but due to testing problems mod(100)

[8] Some results for these sets are invalid (indicated with >), as they hit their 900 second maximum running time.

B.3.1 Increasing number of periods per try

The test results below have been computed using the '2800' computer described in appendix A and the algorithm described in section 2.2.1.

Test Set	UnitMarch_32_bits		UnitMarch_32_bits [VAR][9]	
	Periods	Time	Periods	Time
4blocks	884.3	$13.63^{(8.70)}$	811.9	$13.16^{(8.09)}$
aim-200-2_0-yes1-1	1509.9	$0.49^{(0.57)}$	1633.7	$0.55^{(0.33)}$
aim-200-2_0-yes1-2	39442.4	$13.10^{(14.12)}$	20222.6	$6.90^{(4.04)}$
aim-200-3_4-yes1-1	2321.3	$2.13^{(1.81)}$	1986.6	$1.87^{(1.30)}$
aim-200-3_4-yes1-2	7016.4	$6.42^{(6.51)}$	4451.4	$4.21^{(3.09)}$
aim-200-3_4-yes1-4	3848.1	$3.55^{(2.43)}$	3666.6	$3.48^{(2.38)}$
ais12	635.6	$1.10^{(1.52)}$	943.5	$1.76^{(1.63)}$
bmc-ibm-5	499.5	$9.43^{(8.13)}$	403.7	$8.33^{(4.20)}$
bw_large.c	549.7	$11.76^{(8.82)}$	723.2	$16.48^{(15.77)}$
logistics.a	26281.6	$79.87^{(59.66)}$	21774.3	$69.06^{(57.46)}$
logistics.c	5632.7	$27.30^{(25.90)}$	4555.9	$23.11^{(22.24)}$
par16-1	403.3	$0.32^{(0.44)}$	381.4	$0.32^{(0.34)}$
par16-2	411.9	$0.33^{(0.25)}$	907.6	$0.78^{(0.61)}$
par16-3	669.3	$0.53^{(0.68)}$	290.5	$0.25^{(0.32)}$
par16-4	419.3	$0.33^{(0.33)}$	205.2	$0.18^{(0.17)}$
qg1-08	3772.7	$198.68^{(180.79)}$	5556.4	$309.10^{(203.29)}$
qg2-08	123735.6	$1193.09^{(1126.30)}$	11736.4	$633.15^{(415.77)}$
qg5-11	16.4	$0.57^{(0.75)}$	13.3	$0.62^{(0.61)}$
qg7-13	2855.1	$165.08^{(131.52)}$	2783.7	$203.62^{(102.12)}$
uf250-054	7932.9	$10.26^{(8.71)}$	8325.0	$13.05^{(14.35)}$
uf250-062	4197.7	$5.41^{(4.97)}$	3780.1	$5.85^{(5.87)}$
uf250-071	3487.9	$4.49^{(3.01)}$	4480.1	$6.92^{(4.37)}$
uf250-072	5936.4	$7.82^{(8.85)}$	6341.7	$10.12^{(11.37)}$
uf250-093	2825.1	$3.63^{(2.64)}$	3330.4	$5.54^{(5.09)}$

[9] VAR means that the maximum periods per try is variable. The maximum starts at 10 and each try it is increased by 50%

B.4 Removing bad assignments

The test results below have been computed using the '2800' computer described in appendix A and the algorithm described in section 2.3.

Test Set	UnitMarch_32_bits		UnitMarch_32_bitsk [28,1000][10]		UnitMarch_32_bits [28,100][10]		UnitMarch_32_bits [28,10][10]	
	Periods	Time	Periods	Time	Periods	Time	Periods	Time
4blocks	884.3	$13.63^{(8.70)}$	928.3	$15.70^{(10.79)}$	1727.2	$27.14^{(24.54)}$	13549.2	$253.37^{(195.94)}$
aim-200-2_0-yes1-1	1509.9	$0.49^{(0.57)}$	1200.9	$0.42^{(0.34)}$	639.5	$0.21^{(0.12)}$	4019.9	$1.54^{(1.44)}$
aim-200-2_0-yes1-2	39442.4	$13.10^{(14.12)}$	42967.3	$15.45^{(9.72)}$	12766.5	$4.32^{(2.93)}$	25609.7	$9.94^{(7.42)}$
aim-200-3_4-yes1-1	2321.3	$2.13^{(1.81)}$	2617.3	$2.58^{(1.73)}$	2339.2	$2.15^{(1.75)}$	1799.5	$1.76^{(1.59)}$
aim-200-3_4-yes1-2	7016.4	$6.42^{(6.51)}$	6032.5	$5.92^{(4.49)}$	5342.6	$4.94^{(2.40)}$	4057.3	$3.95^{(3.89)}$
aim-200-3_4-yes1-4	3848.1	$3.55^{(2.43)}$	6231.8	$6.15^{(4.81)}$	8195.9	$7.58^{(6.48)}$	5053.1	$4.93^{(2.95)}$
ais12	635.6	$1.10^{(1.52)}$	502.8	$0.98^{(0.93)}$	281.8	$0.50^{(0.46)}$	731.4	$1.56^{(1.16)}$
bmc-ibm-5	499.5	$9.43^{(8.13)}$	520.8	$10.93^{(9.43)}$	416.0	$8.29^{(8.12)}$	7283.1	$177.16^{(180.30)}$
bw_large.c	549.7	$11.76^{(8.82)}$	566.3	$13.40^{(10.05)}$	498.7	$10.91^{(12.05)}$	637.7	$16.60^{(6.95)}$
logistics.a[11]	26281.6	$79.87^{(59.66)}$	28634.3	$97.60^{(64.26)}$	29419.5	$91.53^{(83.76)}$	56094.1	>209.49
logistics.c[11]	5632.7	$27.30^{(25.90)}$	3053.9	$14.83^{(11.40)}$	9289.4	$46.26^{(46.36)}$	53648.0	>327.46
par16-1	403.3	$0.32^{(0.44)}$	320.7	$0.25^{(0.26)}$	287.1	$0.23^{(0.21)}$	252.6	$0.27^{(0.17)}$
par16-2	411.9	$0.33^{(0.25)}$	411.9	$0.33^{(0.23)}$	734.8	$0.62^{(0.43)}$	657.4	$0.73^{(0.75)}$
par16-3	669.3	$0.53^{(0.68)}$	523.6	$0.41^{(0.36)}$	456.6	$0.37^{(0.29)}$	512.3	$0.56^{(0.48)}$
par16-4	419.3	$0.33^{(0.33)}$	476.7	$0.38^{(0.40)}$	287.1	$0.24^{(0.18)}$	319.5	$0.36^{(0.29)}$
qg1-08[11]	3772.7	$198.68^{(180.79)}$	4678.9	>246.33	2429.1	$131.03^{(177.13)}$	3858.9	>253.66
qg2-08[11]	23735.6	$1193.09^{(1126.30)}$	14052.8	>765.09	14519.8	>804.05	10276.0	>780.13
qg5-11	16.4	$0.57^{(0.75)}$	16.4	$0.59^{(0.73)}$	16.4	$0.57^{(0.72)}$	35.0	$1.39^{(2.35)}$
qg7-13	2855.1	$165.08^{(131.52)}$	3734.7	$216.73^{(149.88)}$	3281.2	$193.17^{(207.35)}$	3250.3	$219.19^{(214.41)}$
uf250-054	7932.9	$10.26^{(8.71)}$	12965.5	$16.81^{(19.05)}$	16332.8	$21.30^{(14.70)}$	33508.9	$47.36^{(58.13)}$
uf250-062	4197.7	$5.41^{(4.97)}$	2180.0	$2.81^{(2.70)}$	3645.1	$4.73^{(3.48)}$	3945.4	$5.58^{(3.65)}$
uf250-071	3487.9	$4.49^{(3.01)}$	5288.9	$6.78^{(3.57)}$	3329.4	$4.30^{(2.74)}$	6592.7	$9.28^{(4.48)}$
uf250-072	5936.4	$7.82^{(8.85)}$	3994.8	$5.27^{(4.49)}$	6008.0	$7.99^{(8.19)}$	5601.1	$8.05^{(6.40)}$
uf250-093	2825.1	$3.63^{(2.64)}$	4507.5	$5.81^{(5.17)}$	2571.8	$3.34^{(3.44)}$	4792.5	$6.76^{(7.94)}$

[10] [x,y] means that the x worst assignments were removed every y periods. While the x worst assignments are removed, the others are randomly flipped with a chance of c%

[11] Some results for these sets are invalid (indicated with >), as they hit their 900 second maximum running time.

B.4.1 Changing kept assignments

The test results below have been computed using the '2800' computer described in appendix A and the algorithm described in section 2.3.1.

Test Set	UnitMarch_32_bits		UnitMarch_32_bits [28,1000] 5%[12]		UnitMarch_32_bits [28,100] 5%[12]		UnitMarch_32_bits [28,10] 5%[12]	
	Periods	Time	Periods	Time	Periods	Time	Periods	Time
4blocks	884.3	$13.63^{(8.70)}$	1047.4	$16.94^{(14.81)}$	2150.7	$35.53^{(37.18)}$	46244.6	$899.84^{(0.03)}$
aim-200-2_0-yes1-1	1509.9	$0.49^{(0.57)}$	1455.4	$0.48^{(0.48)}$	885.5	$0.31^{(0.17)}$	8763.2	$3.50^{(2.57)}$
aim-200-2_0-yes1-2	39442.4	$13.10^{(14.12)}$	32892.5	$11.33^{(13.76)}$	28334.3	$9.90^{(11.45)}$	142500.0	$57.56^{(36.90)}$
aim-200-3_4-yes1-1	2321.3	$2.13^{(1.81)}$	2481.3	$2.34^{(2.51)}$	1913.9	$1.82^{(1.28)}$	2911.9	$2.93^{(2.68)}$
aim-200-3_4-yes1-2	7016.4	$6.42^{(6.51)}$	5614.3	$5.32^{(3.00)}$	8226.8	$7.84^{(9.22)}$	16148.2	$16.34^{(14.40)}$
aim-200-3_4-yes1-4	3848.1	$3.55^{(2.43)}$	4108.9	$3.88^{(2.60)}$	5550.8	$5.29^{(4.49)}$	5694.6	$5.77^{(6.06)}$
ais12	635.6	$1.10^{(1.52)}$	451.6	$0.84^{(0.66)}$	439.5	$0.84^{(0.82)}$	1508.7	$3.41^{(2.42)}$
bmc-ibm-5	499.5	$9.43^{(8.13)}$	509.2	$10.26^{(8.62)}$	584.9	$12.28^{(13.62)}$	7680.2	$196.47^{(188.46)}$
bw_large.c	549.7	$11.76^{(8.82)}$	540.5	$12.25^{(8.47)}$	796.4	$18.66^{(14.44)}$	1293.0	$35.36^{(20.80)}$
logistics.a[13]	26281.6	$79.87^{(59.66)}$	16266.4	$52.42^{(59.67)}$	18580.1	$60.56^{(36.30)}$	214650.0	>849.97
logistics.c[13]	5632.7	$27.30^{(25.90)}$	6131.8	$31.36^{(31.05)}$	20631.2	$107.74^{(84.93)}$	140225.0	>899.96
par16-1	403.3	$0.32^{(0.44)}$	360.0	$0.30^{(0.35)}$	249.1	$0.22^{(0.16)}$	344.6	$0.39^{(0.35)}$
par16-2	411.9	$0.33^{(0.25)}$	411.9	$0.35^{(0.25)}$	667.5	$0.61^{(0.47)}$	817.7	$0.96^{(0.86)}$
par16-3	669.3	$0.53^{(0.68)}$	637.2	$0.53^{(0.56)}$	555.6	$0.48^{(0.33)}$	1187.7	$1.36^{(0.97)}$
par16-4	419.3	$0.33^{(0.33)}$	425.9	$0.35^{(0.34)}$	356.3	$0.31^{(0.21)}$	623.5	$0.71^{(0.46)}$
qg1-08[13]	3772.7	$198.68^{(180.79)}$	2703.8	$149.01^{(111.57)}$	1603.9	$90.35^{(119.10)}$	3878.9	>266.92
qg2-08[13]	123735.6	$1193.09^{(1126.30)}$	10984.4	>624.66	10042.0	>581.65	8215.2	>597.46
qg5-11	16.4	$0.57^{(0.75)}$	16.4	$0.60^{(0.75)}$	16.4	$0.59^{(0.74)}$	24.2	$0.98^{(1.79)}$
qg7-13[13]	2855.1	$165.08^{(131.52)}$	3004.8	$183.77^{(97.62)}$	2704.8	$167.09^{(181.70)}$	4288.5	>293.42
uf250-054	7932.9	$10.26^{(8.71)}$	16016.0	$21.55^{(26.49)}$	18219.5	$24.63^{(26.00)}$	14836.7	$21.26^{(26.73)}$
uf250-062	4197.7	$5.41^{(4.97)}$	1985.7	$2.66^{(3.02)}$	5028.4	$6.82^{(5.48)}$	2928.5	$4.20^{(3.07)}$
uf250-071	3487.9	$4.49^{(3.01)}$	7284.8	$9.74^{(11.68)}$	7469.6	$10.05^{(7.39)}$	22919.8	$32.77^{(24.68)}$
uf250-072	5936.4	$7.82^{(8.85)}$	2860.1	$3.90^{(3.55)}$	4471.8	$6.12^{(6.45)}$	11496.8	$16.76^{(15.21)}$
uf250-093	2825.1	$3.63^{(2.64)}$	3475.2	$4.68^{(4.37)}$	4409.4	$5.97^{(6.55)}$	10845.1	$15.52^{(15.90)}$

[12] [x,y] means that the x worst assignments were removed every y periods. While the x worst assignments are removed, the others are randomly flipped with a probability c

[13] The results for these sets are invalid, as they hit their 900 second maximum running time.

Test Set	UnitMarch_32_bits [28,1000] 1%[14] Periods	UnitMarch_32_bits [28,1000] 1%[14] Time	UnitMarch_32_bits [28,1000] 10%[14] Periods	UnitMarch_32_bits [28,1000] 10%[14] Time
4blocks	959.5	$19.24^{(13.66)}$	1022.0	$16.34^{(12.00)}$
aim-200-2_0-yes1-1	1138.0	$0.41^{(0.31)}$	1566.2	$0.51^{(0.52)}$
aim-200-2_0-yes1-2	27683.9	$12.43^{(10.18)}$	35311.9	$12.04^{(9.94)}$
aim-200-3_4-yes1-1	1855.5	$2.50^{(2.07)}$	2324.3	$2.17^{(1.76)}$
aim-200-3_4-yes1-2	5351.2	$6.34^{(4.03)}$	7661.6	$7.16^{(6.61)}$
aim-200-3_4-yes1-4	3534.9	$4.44^{(2.57)}$	5448.2	$5.12^{(4.05)}$
ais12	439.4	$1.10^{(0.83)}$	478.0	$0.87^{(0.75)}$
bmc-ibm-5	523.6	$14.45^{(11.72)}$	675.4	$13.37^{(14.78)}$
bw_large.c	586.4	$16.35^{(13.36)}$	530.2	$11.79^{(7.93)}$
logistics.a	24179.5	$91.52^{(70.09)}$	24821.8	$78.66^{(52.93)}$
logistics.c	4112.9	$23.36^{(23.54)}$	4946.6	$24.92^{(25.85)}$
par16-1	357.7	$0.31^{(0.36)}$	371.2	$0.30^{(0.37)}$
par16-2	411.9	$0.42^{(0.30)}$	411.9	$0.34^{(0.25)}$
par16-3	626.7	$0.85^{(1.03)}$	693.9	$0.57^{(0.63)}$
par16-4	500.7	$0.46^{(0.55)}$	600.6	$0.49^{(0.61)}$
qg1-08	2678.0	$175.87^{(156.41)}$	4759.8	$262.13^{(174.12)}$
qg2-08[15]	10337.2	>579.89	9829.2	>552.09
qg5-11	16.4	$0.59^{(0.73)}$	16.4	$0.59^{(0.73)}$
qg7-13[15]	5062.2	>303.39	2927.4	$175.99^{(100.17)}$
uf250-054	13495.7	$17.81^{(18.71)}$	9352.7	$12.43^{(11.02)}$
uf250-062	3381.4	$4.44^{(3.93)}$	3663.8	$4.85^{(5.54)}$
uf250-071	4517.5	$5.92^{(5.25)}$	6165.1	$8.11^{(6.22)}$
uf250-072	4297.1	$5.78^{(6.70)}$	6193.9	$8.36^{(11.27)}$
uf250-093	3052.2	$4.01^{(2.73)}$	5918.1	$7.79^{(5.85)}$

[14] [x,y] means that the x worst assignments were removed every y periods. While the x worst assignments are removed, the others are randomly flipped with a probability c

[15] Some results for these sets are invalid (indicated with >), as they hit their 900 second maximum running time.

C Results final version

The test results below have been computed using the '2800' computer described in appendix A and the algorithms described in section 2.1 and 2.2.1.

Test Set	UnitMarch_32_bits		UnitMarch_32_bits [VAR](F=3)[16]	
	Periods	Time	Periods	Time
4blocks	884.3	$13.63^{(8.70)}$	595.0	$13.22^{(7.80)}$
aim-200-2.0-yes1-1	1509.9	$0.49^{(0.57)}$	216.9	$0.09^{(0.07)}$
aim-200-2.0-yes1-2	39442.4	$13.10^{(14.12)}$	3497.9	$1.42^{(1.24)}$
aim-200-3_4-yes1-1	2321.3	$2.13^{(1.81)}$	744.4	$0.77^{(0.78)}$
aim-200-3_4-yes1-2	7016.4	$6.42^{(6.51)}$	2101.5	$2.13^{(1.01)}$
aim-200-3_4-yes1-4	3848.1	$3.55^{(2.43)}$	2759.6	$2.82^{(2.12)}$
ais12	635.6	$1.10^{(1.52)}$	522.0	$1.17^{(1.15)}$
bmc-ibm-5	499.5	$9.43^{(8.13)}$	601.4	$15.87^{(9.57)}$
bw_large.c	549.7	$11.76^{(8.82)}$	522.2	$12.61^{(8.44)}$
logistics.a	26281.6	$79.87^{(59.66)}$	980.6	$3.55^{(1.78)}$
logistics.c	5632.7	$27.30^{(25.90)}$	330.7	$2.06^{(0.80)}$
par16-1	403.3	$0.32^{(0.44)}$	188.6	$0.26^{(0.19)}$
par16-2	411.9	$0.33^{(0.25)}$	910.3	$1.25^{(0.82)}$
par16-3	669.3	$0.53^{(0.68)}$	1487.3	$1.88^{(1.46)}$
par16-4	419.3	$0.33^{(0.33)}$	486.6	$0.65^{(0.68)}$
qg1-08	3772.7	$198.68^{(180.79)}$	1457.9	$90.92^{(72.64)}$
qg2-08	123735.6	$1193.09^{(1126.30)}$	19841.1	$1229.35^{(781.06)}$
qg5-11	16.4	$0.57^{(0.75)}$	48.3	$1.97^{(1.27)}$
qg7-13	2855.1	$165.08^{(131.52)}$	2391.7	$156.45^{(74.09)}$
uf250-054	7932.9	$10.26^{(8.71)}$	4455.9	$6.41^{(3.82)}$
uf250-062	4197.7	$5.41^{(4.97)}$	2917.7	$4.20^{(3.19)}$
uf250-071	3487.9	$4.49^{(3.01)}$	2285.1	$3.27^{(1.83)}$
uf250-072	5936.4	$7.82^{(8.85)}$	2373.9	$3.49^{(1.87)}$
uf250-093	2825.1	$3.63^{(2.64)}$	1758.5	$2.52^{(1.82)}$

[16] VAR means that the maximum periods per try is variable. The maximum starts at 10 and each try it is increased by 50%. In this test the 'New permutation', based on the appearances of variables in unsat clauses is used. (F) that APPEARANCE_FACTOR was defined.