Greedy or Not? Best Improving versus First Improving Stochastic Local Search for MAXSAT

Darrell Whitley, Adele Howe and Doug Hains
Department of Computer Science, Colorado State University, Fort Collins, CO 80523, U.S.A.
{whitley, howe, dhains}@cs.colostate.edu

Abstract

Stochastic local search (SLS) is the dominant paradigm for incomplete SAT and MAXSAT solvers. Early studies on small 3SAT instances found that the use of "best improving" moves did not improve search compared to using an arbitrary "first improving" move. Yet SLS algorithms continue to use best improving moves. We revisit this issue by studying very large random and industrial MAXSAT problems. Because locating best improving moves is more expensive than first improving moves, we designed an "approximate best" improving move algorithm and prove that it is as efficient as first improving move SLS. For industrial problems the first local optima found using best improving moves are statistically significantly better than local optima found using first improving moves. However, this advantage reverses as search continues and algorithms must explore equal moves on plateaus. This reversal appears to be associated with critical variables that are in many clauses and that also yield large improving moves.

Local Search for MAXSAT

Stochastic local search algorithms have successfully solved large random instances as well as industrial instances of both SAT and MAXSAT. "Best improving" local search for MAXSAT identifies all of the improving moves in the Hamming distance 1 neighborhood, then uses the best improving move to update the incumbent solution. In contrast, "first improving" local search arbitrarily selects *the first available* improving move to update the incumbent solution.

Many stochastic local search (SLS) algorithms for MAXSAT can be viewed as having two stages: a greedy stage that uses best improving move local search followed by an exploratory stage that heuristically guides search after plateaus (large regions in the search space in which the majority of moves have the same evaluation) have been encountered (Tompkins, Balint, and Hoos 2011). Gent and Walsh (1992) as well as Schuurmans and Southey (2001) indicated that best improving local search methods are no better than first improving local search, but many current algorithms continue to use best improving local search.

We compare the performance of best improving versus first improving SLS for large uniform randomly generated MAXSAT instances as well as large industrial instances. As in (Tompkins, Balint, and Hoos 2011), we break the search into an initial stage before a local optimum has been found, followed by a second heuristic stage of search. We consider any neighborhood with no improving move to be a "local optimum." Neighborhoods with no improving moves are virtually always on a plateau, and heuristics are used to decide what move to select when there is no improving move.

To support a fair comparison, we introduce a new high fidelity *approximate best* improving move selection method and present proofs showing that this method has $\Theta(1)$ runtime costs per move for all forms of MAXSAT problems. Existing best improving SLS algorithms have runtime costs that can be one or two orders of magnitude slower than using first improving SLS algorithms. Our approximate best improving moves have runtime costs that are the same as using first improving moves. This allows us to ask which strategy yields the best results independent of runtime cost.

In contrast to earlier studies, we find that exploiting best improving moves finds the best solutions at the end of the first stage on industrial problems, but this advantage is lost and reversed in the second stage of search. We explore a number of hypotheses to explain this behavior. The most promising explanation involves the role of critical variables which appear in a large number of clauses in industrial problems. There appears to be an advantage to delaying the exploitation of these variables until the second stage of search.

Understanding SLS for MAXSAT

Let X denote the search space: the set of 0/1 assignments to n Boolean variables. For MAXSAT, the number of clauses, denoted by m, is generally expressed as a function of n and the clause to variable ratio c such that $m = cn = O(n)$. Studies by Gent and Walsh (1992, 1993a, 1993b) analyzed hillclimbing in GSAT (Selman, Levesque, and Mitchell 1992) to determine what contributed to its performance. Gent and Walsh (1993b) examined the influence of greediness (best improving moves) and randomness (first improving moves) on six problems: three SAT encodings of the n-queens problem ($n = 6, 8, 16$) and three random 3SAT problems with clause to variable ratio of 4.3 (for 50, 70 and 100 variables). They found that neither greediness nor randomness were important for GSAT's performance. Gent and Walsh (1993a) divided search into two phases: a short dura-

ID	Instance	n	m
i0	c5-1	200944	540984
i1	fpu	257168	928310
i2	i2c-26	521672	1581471
i3	b15	581064	1712690
i4	rsdKES	707330	1106376
i5	mrisc	844900	2905976
i6	SM-MAIN	870975	3812147
i7	rsd-37	1513544	4909231
i8	wb-4m8s-47	2691648	8517027
i9	mem-ctrl	4426323	15983633

Table 1: Number of variables n and clauses m in selected application instances from the 2012 MAXSAT challenge.

tion of hill climbing followed by a long duration of plateau search. They tested on random 3SAT problems with up to 1000 variables and showed that hill climbing progresses through increasingly lengthening phases in which the incremental improvement declines until a plateau is reached.

Schuurmans and Southey (2001) identified characteristics of local search that can be used to predict behavior. Seven different algorithms were run on MAX-3SAT problems in the phase transition region from SATLIB. They found that effective SAT solvers first find a good local optimum and then explore quickly and broadly to find a solution.

These studies assessed performance on small random 3SAT instances. Do these conclusions extend to large problems and industrial applications where the optimal solution is not known? Our experiments use 10 problems selected at random (with some bias toward not repeating the same types of problems) from the application instances in the SAT 2012 Challenge (http://baldur.iti.kit.edu/SAT-Challenge-2012/) plus three large uniform randomly generated 3SAT instances with 1, 2 and 3 million variables and $c = 4.27$ (denoted by s1, s2, s3). Table 1 lists the number of variables and clauses for the 10 industrial instances.

To obtain a baseline, we ran three incomplete solvers. To control for implementation differences, the three were selected from solvers in UBCSAT (Tompkins and Hoos 2005). GSAT (Selman, Levesque, and Mitchell 1992) was the subject of the Gent and Walsh studies and used best improving moves. *AdaptG^{2}WSAT* (Li, Wei, and Zhang 2007) was one of the best incomplete solvers in the 2007 SAT competition (http://www.satcompetition.org/). *AdaptG^{2}WSAT* follows the two stage paradigm, choosing the best improving move if one exists; otherwise, a heuristic based on Novelty (McAllester, Selman, and Kautz 1997) is used to choose the bit to flip. IRoTS (Smyth, Hoos, and Stützle 2003) performed best in the MAXSAT 2012 Challenge (http://maxsat.ia.udl.cat/results-incomplete/). It either selects the best improving move or randomly flips some number of bits that have not been recently flipped.

Table 2 presents the mean and standard deviations of solutions found over 10 trials, each executed for five minutes of CPU time on a 2.4 GHz Xeon processor with 96GB of RAM. IRoTS maintains a large amount of information about every variable. On large problems, IRoTS struggles because of its high runtime costs, which results in poorer solutions.

ID	GSAT	*AdaptG^{2}WSAT*	IRoTS
i0	662 ± 51	642 ± 60	4853 ± 447
i1	2062 ± 59	2082 ± 78	10443 ± 779
i2	1772 ± 128	1832 ± 84	18197 ± 330
i3	2772 ± 82	3165 ± 132	70063 ± 163
i4	1884 ± 114	1900 ± 159	44310 ± 119
i5	14799 ± 2226	16118 ± 2049	209697 ± 498
i6	7430 ± 289	12124 ± 611	278602 ± 1973
i7	17388 ± 515	563614 ± 2337	613452 ± 588
i8	51654 ± 5197	1316589 ± 1497	1341097 ± 1116
i9	64573 ± 7753	2454452 ± 2115	2468186 ± 1548
s1	9460 ± 494	16919 ± 591	104102 ± 812
s2	34918 ± 112	789594 ± 1887	873326 ± 1385
s3	62442 ± 179	1382446 ± 2598	1400338 ± 3070

Table 2: Mean evaluations (number of unsatisfied clauses) and their standard deviations of solutions found using five minutes of CPU time for UBCSAT versions of GSAT, *AdaptG^{2}WSAT* and IRoTS.

Runtime Costs for Move Selection

To control for differing search cost between best improving and first improving strategies, we developed a technique for efficient approximate best improving move selection. Recall that $m = cn = O(n)$. For MAX-kSAT, the variable k denotes a constant number of variables per clause. We will define a "critical" variable as a variable that appears in more than λ clauses, where λ is a positive integer.

SLS algorithms for MAXSAT use a "$Score$" vector to track the potential contribution of each bit; $Score(i)$ stores partial updates to the evaluation function such that if the current solution is $x \in X$ and x_i is obtained by flipping bit i

$$f(x_i) = f(x) + Score(i).$$

The following results hold for all MAXSAT algorithms that maintain a $Score$ vector. Let U_b be the cost of updating the $Score$ vector after flipping variable b and let $\bar{U}$ be the average cost of updating the $Score$ vector assuming each variable is flipped exactly one time. These will be used to create a foundation for an amortized cost analysis. Let α be a constant that bounds the cost of updating the change in the evaluation function for a single variable in a single clause after one bit is flipped. We next compute the average cost of an update to the $Score$ vector:

Lemma 1. *Assume k is bounded by a constant, and each of the n variables is flipped once: $\bar{U} \leq \alpha ck^2$.*

Proof:

To begin, assume all clauses are of length k and have update costs of exactly α. Then the total number of variable references across all clauses is $mk = nck$. For one clause, each flip causes exactly k updates to the *Score* vector and supporting data structures. Because every variable is flipped once, and there are k bit flips for each clause:

$$\bar{U} = 1/n \sum_{b=1}^{n} U_b \leq 1/n \sum_{j=1}^{m} \alpha k^2 = \alpha k^2 m/n = \alpha ck^2$$

When k and α are upper bounds, we obtain $\bar{U} \leq \alpha c k^2$. $\square$

Theorem 1.

Select constants λ and k such that $\lambda \geq ck$. Assume that no variable appears in more than λ clauses, or if a variable does appear in more than λ clauses it will only be flipped β times during any sequence of n moves. Then the amortized cost per bit flip move associated with updates to the $Score$ vector is $\Theta(1)$ and is bounded by $(\beta+1)\alpha k\lambda$.

Proof:

Assume G variables appear in more than λ clauses and are critical variables that can be flipped at most β times. Collectively (by Lemma 1) the associated runtime cost is bounded as follows:

$$\beta \sum_{j=1}^{G} U_j < \beta \sum_{i=1}^{n} U_i = \beta n \bar{U} \leq \beta n \alpha c k^2$$

For the remainder of the variables, we allow any variable to be flipped at any time until n total moves occur.

The cost of updates associated with one clause is at most αk since the potential contribution of the k variables in the clause must potentially be updated in the $Score$ vector, and α bounds the update cost. If no variable appears in more than λ clauses, collectively this computational cost must be bounded from above by $n\alpha k\lambda$.

Thus on average over any sequence of n bit flips:

$$\frac{\beta \sum_{j=1}^{G} U_j}{n} + \frac{n\alpha k\lambda}{n} \quad \leq \quad (\beta+1)\alpha k\lambda. \quad \square$$

Theorem 1 sets the stage for the following contrary result. We will distinguish between "new" improving moves that have just been updated in the $Score$ vector, and previous "old" improving moves.

Lemma 2.

If there are $O(n)$ old improving moves at time t, in expectation there are $O(n)$ old improving moves at time $t+1$.

Proof: This directly follows from Theorem 1. If the amortized cost of updates to the $Score$ vector is $\Theta(1)$, then the number of changes to the set of improving moves is also bounded by $\Theta(1)$, and on average the number of improving moves decreases by a constant after each bit flip. $\square$

There typically are $\Theta(n)$ improving moves at the beginning of search, and these "old" improving moves will persist. On large problems, we have found that approximately 40 percent of the variables are improving moves when search begins. Such examples can be used to prove that SLS algorithms using best improving moves (e.g., IRoTS, GSAT or *AdaptG^{2}WSAT*) actually have $O(n)$ runtime cost per move because of the way improving moves are stored. Whitley and Chen (2012) point out that one way to control this problem is to use multiple buffers. We operationalize this idea in a new way.

Theorem 2:

Using λ buffers, a form of approximate best improving local search for MAX-kSAT can be implemented with $\Theta(1)$ complexity on average. The search will behave exactly the same as best improving local search if all improving moves satisfy less than λ clauses, or there are no improving moves involving critical variables that have already been flipped β times in the previous $n-1$ moves. If there are multiple "old" moves that satisfy $\geq \lambda$ clauses, an approximate best move may be chosen randomly.

Proof: Theorem 1 accounts for critical variables and amortizes the cost of improving moves. The best "new" improving move can be identified as the $Score$ vector is updated in $\Theta(1)$ time. For $i < \lambda$, buffer i holds improving moves that satisfy i additional unsatisfied clauses. Theorem 1 proves that on average updates to the buffers occur in $\Theta(1)$ time.

If there exist moves that satisfy λ or more clauses, these are placed in buffer λ. If there are multiple old improving moves in buffer λ, an "old" move will be selected randomly from buffer λ. We then choose between the best "new" improving move, or the best "old" (potentially approximate) improving move. This happens in $\Theta(1)$ time. $\square$

ID	First-SLS	Best-13	GSAT
i0	0.09	0.08	0.37
i1	0.12	0.11	1.19
i2	0.18	0.17	1.61
i3	0.38	0.34	5.58
i4	0.16	0.15	1.75
i5	0.42	0.4	6.51
i6	0.50	0.45	12.94
i7	0.71	0.65	17.55
i8	1.35	1.23	56.29
i9	2.65	2.47	218.00
s1	1.27	1.02	26.59
s2	2.83	2.28	108.18
s3	4.53	3.6	243.60

Table 3: CPU time required to reach the first local optimum

Effort Required to Find Local Optima

First improving move SLS might be competitive with best improving move SLS by being faster. Theorem 2 removes this difference. "Approximate best" improving moves can be implemented with the same cost as first improving moves. We use 13 buffers ($4.3*3 \approx 13$); this level of improvement is very rare on random problems. Using as few as four buffers produced almost identical results.

Table 3 presents the mean CPU time required to reach the first local optimum for three algorithms: GSAT from UBCSAT (GSAT), GSAT using first improving moves (First-SLS), and GSAT using best improving moves and 13 buffers (Best-13). The unmodified GSAT has runtime costs that are 1 to 2 orders of magnitude greater than Best-13 on larger problems.

Greedy or Not?

Current state of the art SLS algorithms such as IRoTS and *AdaptG^{2}WSAT* use an implementation of best improving local search that enumerates the set of all possible improving

ID	First-SLS	Best-13	GSAT
i0	30918 ± 115	30093 ± 113	30105 ± 161
i1	36204 ± 270	33285 ± 246	33304 ± 283
i2	57287 ± 313	53615 ± 286	53595 ± 285
i3	74260 ± 338	70464 ± 259	70470 ± 221
i4	52586 ± 160	47483 ± 93	47520 ± 107
i5	118215 ± 969	104237 ± 465	104130 ± 466
i6	135015 ± 779	129252 ± 605	129300 ± 680
i7	244269 ± 475	225977 ± 323	225986 ± 322
i8	414268 ± 439	363405 ± 579	363374 ± 630
i9	609089 ± 1457	578073 ± 1246	577655 ± 967
s1	104123 ± 263	107588 ± 322	107655 ± 253
s2	207942 ± 333	215049 ± 251	214939 ± 400
s3	312925 ± 369	323052 ± 498	322927 ± 451

Table 4: Mean and standard deviations for the evaluations of local optima. GSAT and Best-13 produced better results on the industrial problems, while First-SLS produced better results on the uniform random problems.

moves at each step. Lemma 2 proves that this can result in $O(n)$ runtime cost per move. Furthermore, Gent and Walsh (1993a; 1993b; 1992) did a series of experiments 20 years ago and concluded that first improving local search was just as effective as "greedy" best improving local search. Their studies only used SAT problems with a small number of variables. Do their conclusions hold for larger problems and industrial MAXSAT applications as well as state of the art SLS algorithms?

We looked at search in two stages. The first stage occurs as the search is locating the first local optimum. During this stage, we normally expect best improving move selection to have the most potential impact. In the second stage, a heuristic is often used to decide which bits to flip, but improving moves are selected if they are available.

Stage 1: Finding a First Local Optimum

We assess the choice of move selection in the first stage in two ways. First, are the local optima reached using best improving moves significantly better than for first improving moves? In Table 4, results are given for first improving moves (First-SLS) and best improving moves using 13 buffers (Best-13). For the industrial MAXSAT instances, Best-13 and GSAT yield significantly better local optima, while First-SLS dominates on the random 3SAT instances. For all problems, the differences between First-SLS and Best-13 are statistically significant ($P < 0.0001$) as measured by two tailed t-tests. These results are interesting in three ways. First, the industrial problems and random problems display different patterns. Second, Best-13 is significantly better on industrial instances, in contrast to earlier studies indicating no difference. Third, First-SLS is significantly better on uniform random problems. We examined smaller problems and found that this difference is difficult to detect on problems with less than 1,000 variables, but it is pronounced on problems with 10,000 or more variables.

The local optima provide a starting point for the second stage of search. How much does this starting point matter to overall performance? For the second set of experiments, we used the local optima found by First-SLS and Best-13 to seed the initial solutions used by *AdaptG²WSAT*. Total runtime was limited to five minutes; this was the same time limit as in the MAXSAT 2012 Challenge. We choose *AdaptG²WSAT* because it has performed well in the SAT competitions, it performed the best of all algorithms implemented in UBCSAT on industrial MAXSAT instances (Kroc et al. 2009) and has a clearly defined two stage search strategy. This also allows us to consider different decisions about when to be greedy. Should the algorithm be greedy in stage one, stage two, or in both stages of search? Do these choices make any difference?

The left half of Table 5 shows the results of running *AdaptG²WSAT* starting from solutions found by First-SLS and Best-13 and then using best improving moves (labeled Best) in the second stage of search. Although the local optima found by Best-13 were superior, the results after the second stage using best improving moves suggest that First-SLS provides a better starting point for the second stage in $11/13$ problems with Best in the second stage (all but rsd-KES and 3sat-2m, columns 2 and 3 in Table 5). Two tailed t -tests comparing the approaches on each problem show statistically significant differences ($\alpha < .004$[1]) for three problems: c5-1, rsd-37, and 3sat-3m. The results indicate that the advantage of the best move selection upon arrival at the first local optimum is lost during the remainder of search.

Stage 2: Heuristic Guided Exploration

In the second stage of search, plateaus in the search space present considerable challenges. To address this[2], GSAT introduced sideways moves; Walksat (Selman, Kautz, and Cohen 1994) allowed uphill moves and selected variables randomly from unsatisfied clauses. Not re-flipping the last bit flipped helps prevent cycling, and led to the Novelty family of heuristics which combine a greedy strategy with not selecting recently flipped bits and sometimes pursuing a random walk (McAllester, Selman, and Kautz 1997). *AdaptG²WSAT* included an adaptive noise mechanism. Thus, the second stage of search includes a greedy component, but it is used much less than in the first stage.

To assess the contribution of move strategy in the second stage, we implemented two new variants: First-SLS/First and Best-13/First. First-SLS/First uses first improving moves to reach the first local optimum, and then also uses first improving moves in the second stage of search when there are multiple improving moves; Best-13/First has the same second phase but Best-13 for the first stage. Table 5 (columns 5 and 6) shows the evaluations of the final solutions for these two variants. Best-13/First is worse than First-SLS/First in all but one problem (3sat-1m); two tailed t-tests comparing these two variants show a statistically significant difference ($\alpha = .004$) for $3/13$ problems. Thus, using best improving moves in the first stage can significantly

[1]Because we are running a t-test for each of 13 problems, we apply the Bonferroni adjustment to determine α.

[2]Kautz et al. (Kautz, Sabharwal, and Selman 2009) survey many of the strategies.

ID	First-SLS/Best	Best-13/Best	$P <$	First-SLS/First	Best-13/First	$P <$	$P <$
i0	257 ± 93	665 ± 85	.001	261 ± 72	678 ± 45	.001	.836
i1	1999 ± 21	2029 ± 53	.118	1984 ± 62	2052 ± 50	.015	.473
i2	1802 ± 102	1804 ± 66	.953	1614 ± 68	1642 ± 91	.447	.001
i3	2595 ± 138	2682 ± 149	.191	2587 ± 91	2706 ± 86	.008	.888
i4	1916 ± 232	1906 ± 132	.908	1982 ± 188	1990 ± 146	.914	.492
i5	13689 ± 1726	14617 ± 1847	.261	13650 ± 1795	14671 ± 2007	.246	.961
i6	7222 ± 341	7300 ± 185	.534	6944 ± 248	7130 ± 294	.144	.053
i7	15553 ± 793	17370 ± 546	.001	16039 ± 513	17156 ± 398	.001	.124
i8	41061 ± 3647	45170 ± 4333	.034	43625 ± 3356	45357 ± 3265	.257	.119
i9	36334 ± 2430	37249 ± 2485	.416	34764 ± 2137	34829 ± 2114	.946	.143
s1	9034 ± 552	9248 ± 426	.346	9325 ± 390	9152 ± 652	.482	.192
s2	32363 ± 1299	32047 ± 1714	.649	31117 ± 1001	31759 ± 2025	.385	.028
s3	52365 ± 140	52757 ± 207	.001	52383 ± 210	52796 ± 229	.001	.828

Table 5: Mean and standard deviations of the evaluations of solutions after five minutes of CPU time. The first $P <$ column compares First-SLS/Best and Best-13/Best. The second $P <$ column compares First-SLS/First and Best-13/First; the third compares First-SLS/Best to First-SLS/First.

degrade performance overall even when using first improving moves in the second stage.

Clearly, using First-SLS in the first stage of search yields superior overall performance. Which strategy is best for the second stage? The results are not clear cut. First-SLS/Best-13 has lower mean evaluations in $6/13$ of the problems; none of these are statistically significantly different. Of the seven problems on which First-SLS/First had lower mean evaluations, only one was statistically significantly different at the $.004$ level. Perhaps the strategy used in the second stage does not matter, because there are few improving moves and the improvements are not dramatic.

Explaining the Results

Prior studies showed that it does not seem to matter if local optima are reached using best improving moves compared to first improving moves (Gent and Walsh 1992; Schuurmans and Southey 2001). We expected to find the same. In contrast to expectations, using first improving move SLS on large industrial problems can result in better solutions than using best improving moves. Although the initial local optima have worse evaluations for $11/13$ problems, the final solutions derived from local optima found by first improving moves were better than those found by using best improving moves. Clearly, this is not explained by runtime cost, since our approximate best improving move selection is just as fast as using first improving moves.

Differences in the problem sets may be a factor. When we looked at performance on a range of MAX-3SAT problems from 100 to 1,000,000 variables, we also did not detect a significant difference on smaller problems ($< 1,000$ variables). But for larger problems ($> 10,000$) first improving move finds significantly better solutions and the difference increases as n increases.

Could it be that the local optima found by First-SLS are actually closer to the best solution? To test this, we ran *AdaptG²WSAT* for 24 hours on each problem to find a "best" solution and then computed the Hamming distance between the best solution and the local optima found by our

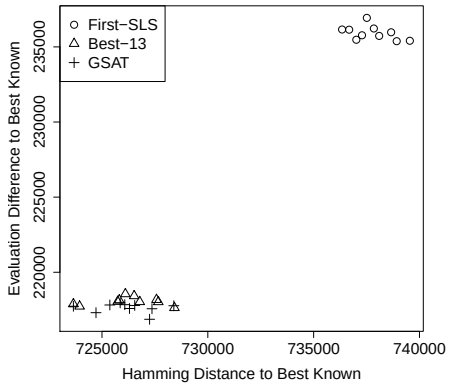

Figure 1: Evaluation difference versus Hamming distance from first local optima to best found solution for local search variants on rsd-37.

descent variants. Plots of Hamming distance versus evaluation show distinctive patterns. On the smaller application problems (the four smallest problems in our set), First-SLS produces solutions with worse evaluation, but which are closer to the best found. On the six larger problems, the solutions produced by First-SLS are both worse and further away from the best found solution, e.g., rsd-37 as shown in Figure 1. For all problems, the variants of first improving appear to cluster with similar evaluations and Hamming distances. Clearly, first improving moves are taking search to a different part of the space; however, differences in Hamming distance do not explain the superiority of the local optima as starting points for exploration.

To better understand how and when First-SLS gained an advantage over best improving strategies, we also plotted the improvement over time for many runs of First-SLS/Best and Best-13/Best. The pattern observed in Figure 2 was found in all runs and all problem instances. This pattern occurs

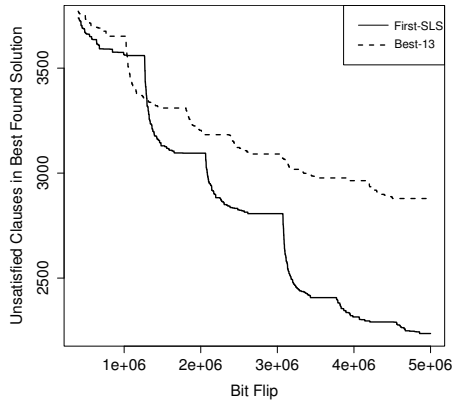

Figure 2: Snapshot of improvement over time after initial descent using first improving and best improving moves for problem C5-1 during the second stage of search.

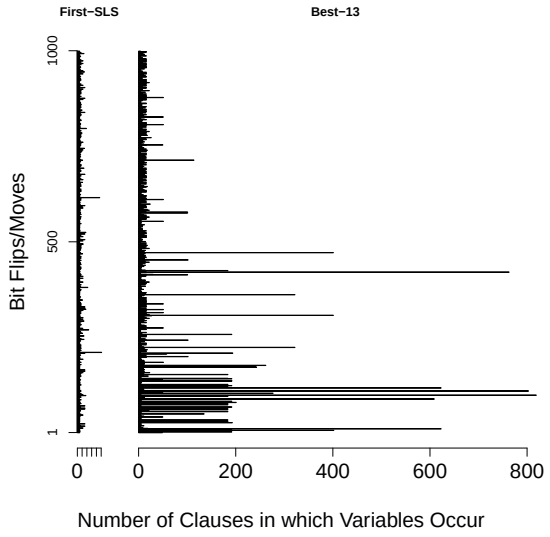

Figure 3: Number of clauses in which variables occur is plotted against the first 1000 bits flipped by first (left) and best (right) improving moves on problem C5-1.

in the second stage of search. When First-SLS is used during the initial phase of search, the second phase is able to periodically find bit flips that result in rapid progress. SLS algorithms that use best improving moves during the first phase also have periodic events of rapid improvement in the second stage of search, but these events are not as dramatic and result in less overall improvement.

Our best hypothesis is that using first improving moves during the first phase of the search does not fix the critical variables that appear in many clauses; in contrast, best improving moves is more likely to fix such variables as they are more likely to yield the greatest improvement. On a 1 million variable problem with 4.3 million clauses, we found there are typically 400,000 improving moves of which about $1/2$ satisfied 1 additional clause, $1/4$ satisfied 2 additional clauses, $1/8$ satisfied 3 additional clauses, etc. After the first few hundred bit flips, improving moves that satisfy more than 3 clauses have almost entirely vanished, but 90% or more of the improving moves that satisfy 1 to 3 clauses persist. This pattern was similar for both randomly generated problems and industrial problems. The use of best improving moves leads to a pattern consistent with a model of local search developed in (Gent and Walsh 1993a) where hill climbing exhibits increasingly longer phases of decreasing incremental improvement.

Why does delaying the exploitation of critical variables result in bursts of rapid progress in the second stage of search? If i and j appear together in a clause in a uniform random problem, the chances that they appear together again is very small and decreases as n becomes larger. However, if i and j appear together in an industrial problem, i and j might co-occur 4 times due to the way that general Boolean satisfiability problems are converted to MAXSAT problems (Cormen et al. 2001). In addition, critical variables co-occur more often in industrial problems. Thus, when a critical variable is flipped, this can result in an unusually large number of changes in the Score vector, even if the move is a non-improving "equal" move.

When first improving moves are utilized, improving moves associated with critical variables are lost in a sea of moves that satisfy 1 or 2 or 3 variables; thus most critical variables are not exploited in the initial phase of search. But best improving search *does* exploit these critical variables. This is illustrated in Figure 3 which shows that on problem C5-1 many critical variables appearing in 100's of clauses are flipped in the first 500 moves by Best-13; no variable flipped by First-SLS appears in more than 50 clauses, and only 1 appears in more than 20 clauses. By not prematurely fixing critical variables, First-SLS is leaving the second stage of search with untapped opportunities to change many clauses at once, and thus change the direction of the search (and the Score vector) in a dramatic fashion.

Conclusions and Future Work

Our study has re-visited the issue of whether greedy search is influential for the first phase of search. Our evidence suggests that not only does the move selection strategy make a difference to performance, but that using first improving moves results in better solutions on large random and industrial MAXSAT problems. We studied 10 problems in detail in this paper; we have extended this study to 18 additional problems. The performance of First-SLS/Best is similar on all 28 problems (better on 24/28 compared to 9/10). Future work should also further investigate the role that critical variables play in industrial MAXSAT instances.

Acknowledgments

This research was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, grant number FA9550-11-1-0088. The U.S. Government is authorized to reproduce and distribute reprints notwithstanding any copyright notation thereon.

References

Cormen, T.; Leiserson, C.; Rivest, R.; and Stein, C. 2001. *Introduction to Algorithms*. MIT press.

Gent, I. P., and Walsh, T. 1992. The enigma of SAT hill-climbing procedures. Technical Report TR 605, University of Edinburgh, Department of Artificial Intelligence.

Gent, I. P., and Walsh, T. 1993a. An empirical analysis of search in GSAT. *Journal of Artificial Intelligence Research* 1:47–59.

Gent, I. P., and Walsh, T. 1993b. Towards an understanding of hill-climbing procedures for SAT. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 28–33. MIT Press.

Kautz, H.; Sabharwal, A.; and Selman, B. 2009. Incomplete algorithms. *Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications* 185:185–204.

Kroc, L.; Sabharwal, A.; Gomes, C.; and Selman, B. 2009. Integrating systematic and local search paradigms: A new strategy for MaxSAT. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence*, 544–551.

Li, C.; Wei, W.; and Zhang, H. 2007. Combining adaptive noise and look-ahead in local search for SAT. *Theory and Applications of Satisfiability Testing–SAT 2007* 121–133.

McAllester, D.; Selman, B.; and Kautz, H. 1997. Evidence for invariants in local search. In *Proceedings of the National Conference on Artificial Intelligence*, 321–326.

Schuurmans, D., and Southey, F. 2001. Local search characteristics of incomplete SAT procedures. *Artificial Intelligence* 132(2):121–150.

Selman, B.; Kautz, H.; and Cohen, B. 1994. Noise strategies for improving local search. In *Proceedings of the National Conference on Artificial Intelligence*, 337–337.

Selman, B.; Levesque, H.; and Mitchell, D. 1992. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial intelligence*, 440–446.

Smyth, K.; Hoos, H. H.; and Stützle, T. 2003. Iterated robust tabu search for MAX-SAT. In *Proceedings of the 16th Conference of the Canadian Society for Computational Studies of Intelligence*, 129–144.

Tompkins, D., and Hoos, H. 2005. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In *Revised Selected Papers from the Seventh International Conference on Theory and Applications of Satisfiability Testing*, volume 3542 of *Lecture Notes in Computer Science*, 306–320.

Tompkins, D.; Balint, A.; and Hoos, H. 2011. Captain Jack: New variable selection heuristics in local search for SAT. *Theory and Applications of Satisfiability Testing-SAT 2011* 302–316.

Whitley, D., and Chen, W. 2012. Constant Time Steepest Ascent Local Search with Statistical Lookahead for NK-Landscapes. In *GECCO '12: Proc. of the annual conference on Genetic and Evolutionary Computation Conference*.