Accelerating Random Walks

Wei Wei and Bart Selman

Dept. of Computer Science
Cornell University
Ithaca, NY 14853

Abstract. In recent years, there has been much research on local search techniques for solving constraint satisfaction problems, including Boolean satisfiability problems. Some of the most successful procedures combine a form of random walk with a greedy bias. These procedures are quite effective in a number of problem domains, for example, constraint-based planning and scheduling, graph coloring, and hard random problem instances. However, in other structured domains, backtrack-style procedures are often more effective. We introduce a technique that leads to significant speedups of random walk style procedures on structured problem domains. Our method identifies long range dependencies among variables in the underlying problem instance. Such dependencies are made explicit by adding new problem constraints. These new constraints can be derived efficiently, and, literally, "accelerate" the Random Walk search process. We provide a formal analysis of our approach and an empirical validation on a recent benchmark collection of hardware verification problems.

1 Introduction

Local search style methods have become a viable alternative to constructive backtrack style methods for solving constraint satisfaction and Boolean satisfiability problems. Local search techniques were originally introduced to find good approximate solutions to optimization problems [14]. In subsequent years, many refinements have been introduced such as simulated annealing and tabu search. More recently, local search methods have also been used for solving decision style problems, in particular Boolean satisfiability (SAT) problems. We will restrict most of our discussion to the SAT domain. However, many of the insights should carry over to the richer domain of general constraint satisfaction problems.

The first local search methods for SAT, such as GSAT, were based on standard greedy hill-climbing search [26, 10, 8], inspired by heuristic repair techniques as studied for optimization [17]. A major improvement upon these algorithms was obtained by building the search around a so-called random walk strategy, which led to WalkSat and related methods [24]. These algorithms combine a Random Walk search strategy with a greedy bias towards assignments with more satisfied clauses. A random walk procedure for satisfiability is a deceptively simple search technique. Such a procedure starts with a random truth assignment. Assuming this randomly guessed assignment does not already satisfy the formula, one selects one of the unsatisfied clauses at random, and flips the truth assignment of one of the variables in that clause. This will cause the clause to become satisfied but, of course, one or more other clauses may become unsatisfied. Such flips are repeated until one reaches an assignment that satisfies all

clauses or until a pre-defined maximum number of flips is made. This simple strategy can be suprisingly effective. In fact, Papadimitriou [19] showed that a pure (unbiased) random walk on an arbritary satisfiable 2SAT formula will reach a satisfying assignment in $O(N^2)$ flips (with probably going to 1). More recently, Schoening [22] showed that a series of short unbiased random walks on a 3-SAT problem will find a satisfying assignment in $O(1.334^N)$ flips (assuming such an assignment exists), much better than $O(2^N)$ for an exhaustive check of all assignments.

In the next section, we will discuss how one can introduce a greedy bias in the Random Walk strategy for SAT, leading to the WalkSat algorithm. WalkSat has been shown to be highly effective on a range of problem domains, such as hard random k-SAT problems, logistics planning formulas, graph coloring, and circuit synthesis problems [24, 12]. However, on highly structured formulas, with many dependent variables, as arise in, for example, hardware and software verification, the WalkSat procedure is less effective. In fact, on such domains, backtrack style procedures, such as the Davis-Putnam-Logemann-Loveland (DPLL) [2, 3] procedure are currently more effective. One of the key obstacles in terms of WalkSat's performance on highly structured problems is the fact that a random walk will take at least order N^2 flips to propagate dependencies among variables. On the other hand, unit-propagation, as used in DPLL methods, handles such dependencies in linear time. With thousands or ten of thousands of variables in a formula, this difference in performance puts local search style methods at a distinct practical disadvantage. Moreover, there are also chains of simple ternary clauses that require exponential time for a random walk strategy, yet can be solved in linear time using unit propagation [21].

We will consider such dependency chains in detail and provide a rigorous analysis of the behavior of a pure random walk strategy on such formulas. This analysis suggest a mechanism for dealing more effectively with long range dependencies. In particular, we will show how by introducing certain implied clauses, capturing the long range dependencies, one can significantly "accelerate" random walks on such formulas. We show how the implied clauses can often reduce quadratic convergence time down to near linear, and, in other cases, reduce exponential time convergence of the random walk to polytime convergence. Moreover, the implied clauses can be derived efficiently.

The speedups obtained on the chain-like formulas are encouraging but leave open the question of whether such an approach would also work on practical instances. We therefore validated our approach on a range of highly structured test problems from hardware verification applications [28, 29]. Our experiments show that after adding certain implied dependencies, WalkSat's overall performance improves significantly, even when taking into account the time for computing the dependencies. This work provides at least a partial response to the challenge in [25] on how to improve local search methods to deal with dependent variables in highly structured SAT problems.

We would also like to stress the benefit from the particular methodology followed in this work. By first analyzing special classes of formulas, which bring out "extreme" behavior of the random walk process, we obtain valuable insights to remedy this behavior. These insights are subsequently used to improve performance on more general formulas. Given the intractability of directly analyzing local search behavior on general classes of formulas, we believe the use of special
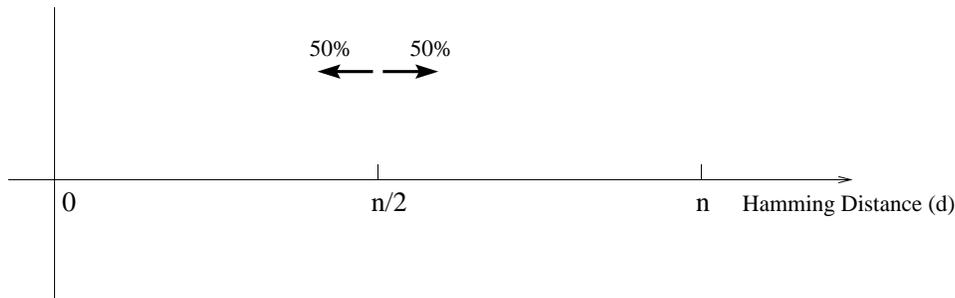
50%   50%

0   n/2   n   Hamming Distance (d)

Fig. 1. A Random Walk.

restricted classes is highly beneficial. There are no doubt many other forms of problem structure that could be studied this way.

The paper is structured as follows. In section 2, we first review Random Walk strategies — biased and unbiased — for SAT. In section 3, we give results on special classes of chain formulas and provide a formal analysis. In section 4, we give an empirical evaluation of our approach. Section 5 contains conclusions and future directions.

2 Random Walk Strategies for SAT

To gain some further insight into the behavior of random walk strategies for SAT, let us briefly consider the very elegant argument introduced by Papadimitriou showing polytime behavior on 2SAT. Consider a satisfiable 2SAT formula F on N variables. Let T denote a satisfying assignment of F. The random walk procedure starts with a random truth assignment, T'. On average, this truth assignment will differ from T on the assignment of $N/2$ letters. Now, consider an unsatisfied clause in F (if no such clause exists, then T' is a satisfying assignment). Without loss of generality, we assume that the unsatisfied clause is of the form $(a \vee \neg b)$. Since this clause is unsatisfied, T' must assign both literals in the clause to False (which means that a is assigned False and b True). Also, the satisfying assignment T is such that at least one of the literals in the clause is assigned to True. Now, randomly select a variable in the clause and flip its truth value in T'. Since we have only two variables in the clause, we have at least a 50% chance of selecting the variable corresponding to the literal set to True in T. (Note that if T satisfied exactly one literal in the clause, we will select that literal with 0.5 probability. If T satisfies both literals, we select the "right" literal with probability 1.0.) It follows that with at least 50% chance, the Hamming distance of our new truth assignment to T will be reduced by 1, and with at most 50% chance, we will have picked the wrong variable and will have increased the Hamming distance to the satisfying assignment. Papadimitriou now appeals to a basic result in the theory of random walks. Consider a one dimensional "random walker" which starts at a given location and takes L steps, where each step is either one unit to the left or one unit to the right, each with probability 0.5 (also called a "drunkards walk"). It can be shown that after L^2 steps, such a walker will on average travel a distance of L units from its starting point. Given that the random walk starts a distance $N/2$ from the satisfying truth assignment T, after order N^2 steps, the walk will hit a satisfying assignment, with probability going to one. Note that

although the walk may at first wander away from the satisfying assignment, it will "bounce off" the reflecting barrier at distance N. Also, our analysis is worst-case, *i.e.,* it holds for *any* satisfiable 2SAT formula.

Procedure RW
repeat
      c:= an unsatisfied clause chosen at random
      x:= a variable in c chosen at random
      flip the value of x;
until a satisfying assignment is found.

Procedure RWF
repeat
      c:= an unsatisfied clause chosen at random
      if there exists a variable x in c with break value = 0
            flip the value of x (freebie move)
      else
            x:= a variable in c chosen at random from c;
            flip the value of x
until a satisfying assignment is found.

Procedure WalkSat
repeat
      c:= an unsatisfied clause chosen at random
      if there exists a variable x in c with break value = 0
            flip the value of x (freebie move)
      else
            with probability p
                x:= a variable in c chosen at random;
                flip the value of x
            with probability (1-p)
                x:= a variable in c with smallest break value
                flip the value of x
until a satisfying assignment is found.

Fig. 2. The main loop of the Random Walk (RW), Random Walk with freebie (RWF), and WalkSat procedures.

It is instructive to consider what happens on a 3SAT formula, *i.e.*, a conjunctive normal form formula with 3 literals per clauses. In this case, when flipping a variable selected at random from an unsatisfied clause, we may only have 1/3 chance of fixing the "correct" variable (assuming our satisfying assignment satisfies exactly one literal in each clause). This leads to a random walk heavily biased away from the solution under consideration. The theory of random walks tells us that reaching the satisfying assignment under such a bias would take an exponential number of flips. And, in fact, in practice we indeed see that a pure random walk on a hard random 3SAT formula performs very poorly.

However, we can try to counter this "negative" bias by considering the gradient in the overall objective function we are trying to minimimize. The idea is that the gradient may provide the random walk with some additional information "pointing" towards the solution, and thus towards the right variable to flip. In SAT, we want to minimize is the number of unsatisfied clauses. So, in selecting

the variable to flip in an unsatisfied clause, we can bias our selection towards a variable that leads to the greatest decrease in the overall number of unsatisfied clauses. Introducing a bias of this form leads us to the WalkSat algorithm and its variants [24, 15].

For reasons of efficiency, WalkSat uses a bias defined by the so-called "break value" of a variable. Given a formula and a truth assignment T, the break value of a variable, x, is defined by the number of clauses that are satisfied by T but become unsatisfied (are "broken") when the truth value of x is changed. In practice, this bias is a good measure of the overall change in number of unsatisfied clauses when x is "flipped".

WalkSat boosts its search by interleaving purely random walk moves and greedily biased moves. The number of random walk moves is controlled by a parameter p. That is, with probability p, we select a random variable from the chosen unsatisfied clause, and with probability $1 - p$, we select a variable in the clause with the minimal break value (ties are broken randomly). In practice, one can often identify a (near) optimal value for p for a given class of formulas. (For example, $p = 0.55$ for hard random 3SAT formulas, and $p = 0.3$ for planning problems.)

As a final refinement, WalkSat incorporates one other important feature to speed up its search: when there is a variable in the randomly selected unsatisfied clauses with break value zero, that variable is flipped immediately. Note that such a flip will increase the total number of satisfied clauses by at least one. These flips are called "freebies".

In order to study the introduction of a greedy bias in its minimal form, we will also consider a Random Walk with just "freebie" moves added. This is equivalent to running WalkSat with noise parameter $p = 1.0$. Figure 2 gives the main loop of the Random Walk (RW) procedure, the Random Walk with freebie (RWF) procedure, and the WalkSat procedure.[1]

3 Chain Formulas and Theoretical Underpinning

In order to obtain a good understanding of the behavior of pure Random Walk and biased Random Walk strategies on SAT instances, we now consider several families of formulas which demonstrate extreme properties of random walk style algorithms. We elucidate these properties by introducing long chains of dependencies between variables. We consider 2SAT and 3SAT problems. Our 3SAT formulas are motivated by formulas introduced by Prestwich [21].

3.1 Binary Chains

In the previous section, we derived a quadratic upperbound for the number of flips for solving a 2SAT problem using a unbiased Random Walk strategy. In practice, on for example, randomly generated 2SAT problems, Random Walk may take far fewer than N^2 flips. How would one construct a 2CNF formula that exhibits the worst case quadratic behavior? To obtain such a formula we need to create a truly unbiased random walk. (A truly unbiased random walk takes $\Theta(N^2)$ flips to travel a distance N.) As we argued above, when flipping a

---

[1] The WalkSat program [35] performs RW when option `-random` is selected. When selecting option `-noise 100 100` (*i.e.*, p = 1.0), we obtain RWF.
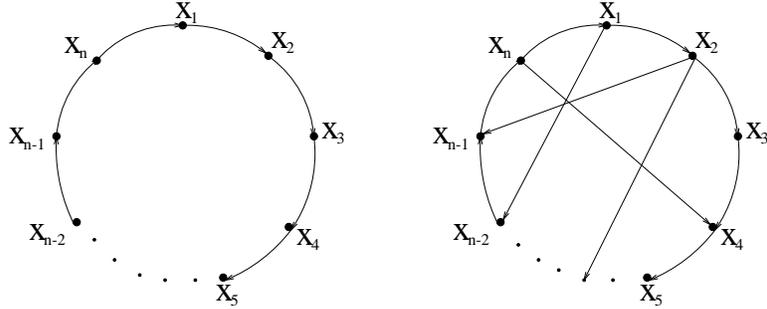
Fig. 3. The left panel shows a graph of a binary chain formula, F_{2chain} on n variables. The right panel shows the formula after adding a number of random implied clauses.

random variable in a binary clause, we have at least 50% chance of "fixing" the correct variable (on a path to a satisfying assignment). We need to ensure that we have exactly 50% chance of selecting the right variable to flip. We do this by constructing a formula such that a satisfying assignment of the formula satisfies exactly one literal (a variable or its negation) in each clause.

This can be achieved by considering the following 2CNF formula, F_{2chain}, consisting of a chain of logical implications: $(x_1 \rightarrow x_2) \wedge (x_2 \rightarrow x_3) \wedge \ldots \wedge (x_N \rightarrow x_1)$. See Figure 3, left panel. This formula has two assignments, namely the all True assignment and the all False assignment. Each clause is of the form $\neg x_i \vee x_{i+1}$). So, each of the two satisfying assignment satisfies exactly one literal in each clause. We therefore obtain a truly unbiased random walk, and we obtain the following result.

Theorem 1. *The RW procedure takes $\Theta(N^2)$ to find a satisfying assignment of F_{2chain}.*

In contrast, DPLL's unit propagation mechanism finds an assignment for F_{2chain} in linear time. Moreover, our experiments show that adding a greedy bias to our random walk does not not help in this case: both the RWF and the WalkSat procedure take $\Theta(N^2)$ flips to reach a satisfying assignment on these formulas.[2]

3.2 Speeding Up Random Walks on Binary Chains

Given the difference in performance between DPLL and random walk style algorithms on the 2SAT chain formulas, the question becomes whether we can somehow speed-up the random walk process on such formulas to perform closer to DPLL.

Intuitively, the random walk process needs to line up the truth values along the chain, but can only do so by literally "wandering" back and forth based on

---

[2] In this paper, we derive rigorous results for the unbiased random walk procedure. Our results for biased random walk strategies are empirical. The reason for this is that the tools for a rigorous analysis of greedy local search methods are generally too weak to obtain meaningful upper- and lower-bounds on run time, as is apparent from the relative paucity of rigorous run time results for methods such as simulated annealing and tabu search (expect for general convergence results).

local inconsistencies detected in the chain. One possible remedy is to add clauses that capture longer range dependencies between variables more directly.

We therefore consider adding clauses of the form $x_i \rightarrow x_j$, with i and $j \in \{1, 2, \cdots, n\}$, chosen uniformly at random. A binary chain with redundancy (BCR) formula is composed of all clauses from binary chain F_{2chain} and a fraction of the redundant clauses, generated independently at random. Since any redundant clause is implied by the binary chain, it follows that any BCR formula is equivalent to the chain itself. We define the redundancy rate of a BCR formula as the number of redundant clauses divided by the length of its base binary chain. The right hand side panel of Figure 3 shows a binary chain formula with added random redundancies.[3]

Perhaps somewhat suprisingly, the implied "long-range" clauses do not make any difference for the performance of an unbiased random walk. The reason for this is that the underlying random walk remains completely balanced (*i.e.*, 50% chance of going in either direction). This follows from the fact that each of the added clauses again has exactly one literal satisfied in the two satisfying assignments.

However, the situation changes dramatically for the biased random walk case. The right panel of Figure 4 gives the scaling for RWF for different levels of redundancies. We see that the scaling improves dramatically with increased levels of implied long range clauses. More specifically, with a redundancy rate of 0, we have a scaling of $\Theta(N^2)$, wich improves to $\Theta(N^{1.2})$ for a redundancy rate of 2. (Regression fits with MatLab.) The right panel shows that we get a further improvement when using WalkSat. WalkSat scales as $\Theta(N^{1.1})$ (redundancy rate of 2). So, we note that the performance of WalkSat becomes quite close to that of unit propagation. Interestingly, when we further increase the level of implied clauses, the performance of biased random walk procedures starts to decrease again. See Figure 5. We will encounter a similar phenomenon when considering our benchmark problems from hardware verification.

The clauses added to the binary chain can be understood as results of a series of resolutions. Therefore, we have shown for this basic structure — which brings out the worst case behavior in random walks on 2SAT problems — that the added long distance resolvents, coupled with a greedy bias in the local search algorithm, effectively reduces the run time required to find a solution. Of course, for solving a single 2SAT chain formula in isolation, the process of adding long range resolvents would not make much sense, since the formulas are easily solvable using just unit-propagation. However, if such chains are embedded in much larger formulas, then speeding up the random walk process can be useful because it may speed up the overall solution process by propagating dependency information much more quickly. In fact, we will see in our experimental validation section that this is indeed the case on large-scale practical formulas from the hardware verification domain.

---

[3] For readers familiar with the recent work by Watts and Strogatz [32] on 'small world' graphs, we note the resemblence of our 2SAT chain with redundancies and small world graphs. In fact, the small world topology where long range and short range interactions are combined provided an impetus for this work. We are currently exploring further connections to computational properties of small world graphs. See also [13, 31].
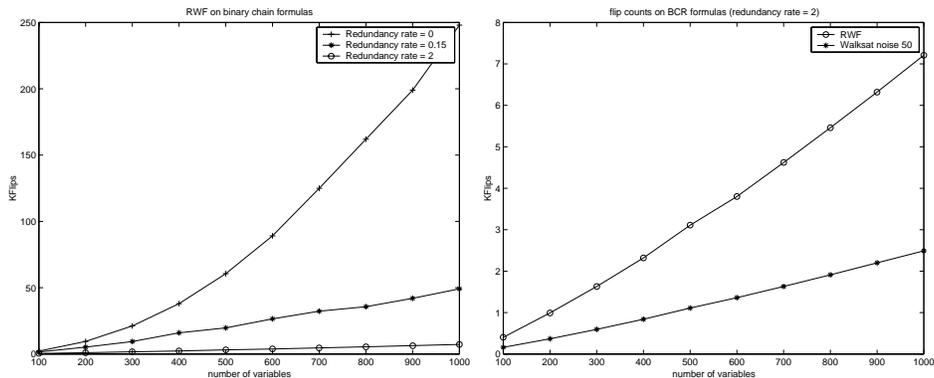
Fig. 4. The left panel shows the number of flips vs. the number of variables in binary chains of different sizes. The right panel shows the curves for RWF and WalkSat algorithms when the redundancy rate is set at 2.
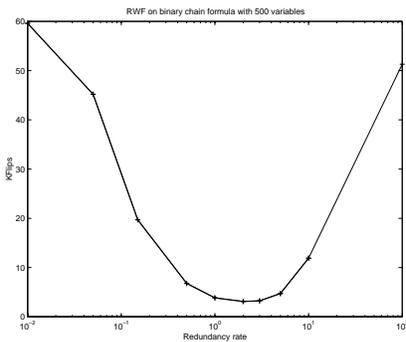

Fig. 5. This figure shows the number of flips RWF makes on the 500-variable binary chain formulas with different redundancy rates.

3.3 Ternary Chains

We now consider another class of formulas, involving ternary clauses. As discussed in Section 2, on ternary clauses the Random Walk strategy becomes biased away from the solution, at least in the worst case. When faced with such a bias, a Random Walk would require an exponential number of flips to reach a satisfying assignment. This effect is quite dramatic. For example, in Figure 1, a random walk with just a 0.1% bias in the wrong direction, i.e., 49.9% to the left and 50.1% to the right will take exponential time to reach the origin. We will give a concrete family of chain-like formulas that exhibit such exponential worst-case behavior. However, on the positive side, we will also show how a quite similar family of 3CNF formulas exhibits polynomial time behavior. *By considering the differences between the intractable and the tractable structures, we obtain new insights into methods for speeding up random walk techniques.* To the best of our knowledge, our results are the first rigorous results identifying a tractable class of 3CNF problems for a random walk style local search method.

In our analysis, we consider formulas inspired by ternary chain formulas first introduced by Prestwich [21]. Prestwich provided empirical results on such for-
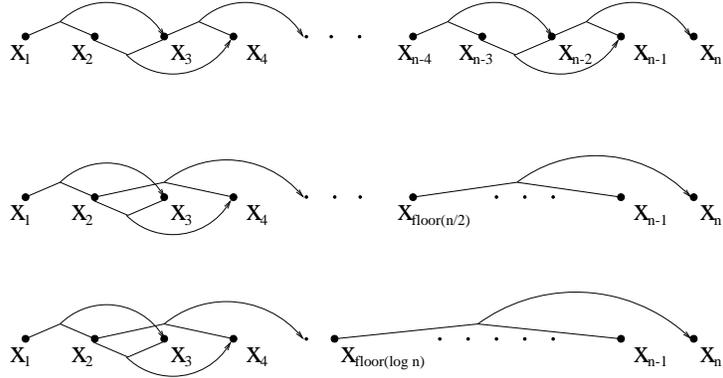
Fig. 6. These three diagrams illustrate our ternary chain formulas. The top panel shows $F_{3chain,i-2}$ which has local links in the form of $x_{i-2} \wedge x_{i-1} \to x_i$ only. The middle panel gives $F_{3chain,\lfloor \frac{i}{2} \rfloor}$ which has longer links of the form $x_{\lfloor \frac{i}{2} \rfloor} \wedge x_{i-1} \to x_i$. The bottom panel shows $F_{3chain,\lfloor \log i \rfloor}$ with even longer links of the form of $x_{\lfloor \log i \rfloor} \wedge x_{i-1} \to x_i$.

mulas for the WalkSat procedure. Our analysis extends his observations, and provides the first rigorous proofs for the convergence rates of unbiased Random Walk on these formulas. Our results also show that on these formulas the difference between biased and unbiased walks is minimal.

Let low(i) be a function that maps i, with $3 \leq i \leq n$, into an integer in the range from 1 to $i - 2$. We consider ternary chain formulas, $F_{3chain,low(i)}$, on variables x_1 through x_n consisting of a conjunction of the following clauses:

$$x_1$$
$$x_2$$
$$x_1 \wedge x_2 \to x_3$$
$$\vdots$$
$$x_{\text{low}(i)} \wedge x_{i-1} \to x_i$$
$$\vdots$$
$$x_{\text{low}(n)} \wedge x_{n-1} \to x_n$$

Depending on our choice of the function low(i), we obtain different classes of formulas. For example, with low(i)$= i - 2$, we get the formula

$$x_1 \wedge x_2 \wedge (x_1 \wedge x_2 \to x_3) \wedge (x_2 \wedge x_3 \to x_4) \wedge \ldots \wedge (x_{n-2} \wedge x_{n-1} \to x_n).$$

Note that the function low(i) determines how the implications in the chain are linked together. We will consider three different ways of linking the left hand side variable in each implication with the earlier variables in the chain. Our choices are:

a) Short range connections: low(i)$= i - 2$.
b) Medium range connections: low(i)$= \lfloor \frac{i}{2} \rfloor$.
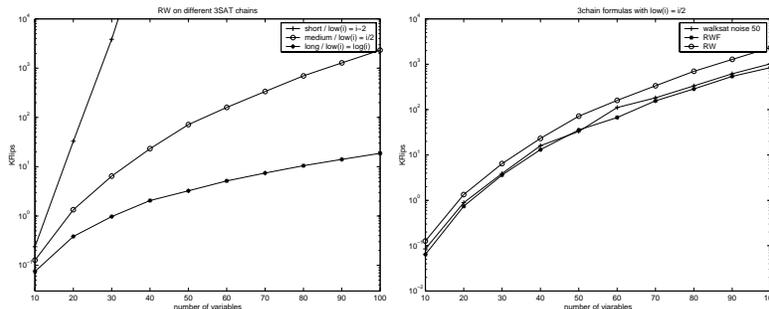c) Long range connections: low(i)$= \lfloor \log i \rfloor$.

Fig. 7. RW on tenary chains (left). RW, RWF, and WalkSat on ternary chain with low(i)= $\lfloor \frac{i}{2} \rfloor$ (right).

Figure 6 illustrates each of these formula classes. We refer to these classes of formulas by given low(i) as a subscript. We thus obtain the classes $F_{3chain,i-2}$, $F_{3chain,\lfloor \frac{i}{2} \rfloor}$, and $F_{3chain,\lfloor \log i \rfloor}$.

It is clear that each of our ternary chain formulas is trivially solvable by unit propagation since the effect of the first two unit clauses x_1 and x_2 directly propagates through the chain of implications. The only satisfying assignment is the all True assignment.[4] However, it is far from clear how long a Random Walk would take to find the satisfying assignment, when starting from a random initial assignment. This brings us to our main formal results.

We found that the convergence rate of RW differs dramatically between the three formula classes, *ranging from exponential on $F_{3chain,i-2}$ (short range connections), to quasi-polynomial on $F_{3chain,\lfloor \frac{i}{2} \rfloor}$ (medium range connections), and finally to polynomial on $F_{3chain,\lfloor \log i \rfloor}$ (long range connections).*

The tractable cases are especially encouraging, since it has been widely believed that an unbiased RW can hardly solve any 3-SAT formulas in less than exponential time (unless such formulas have an exponential number of assignments). Consider RW on any of our ternary chain formulas after selecting an unsatisfied clause. The procedure now has only a probability of 1/3 of selecting the right variable to fix. To see this, consider a clause in one of our chain formulas that is not satisfied by the current truth assignment, *e.g.*, $x_{low(i)} \wedge x_{i-1} \rightarrow x_i$. This means that the current truth assignment must assign $x_{low(i)}$ and x_{i-1} to True and x_i to False. The "correct" flip to make would be to assign x_i to True. However, with probability 2/3, the RW procedure will flip either $x_{low(i)}$ or x_{i-1} to False. Thus, strongly biasing the walk away from the unique all True solution. This general argument might lead one to conclude that RW should take exponential time on all three of our ternary chain formula classes. However, a much more refined analysis, which also takes into account how variables are shared among clauses and the order in which the clauses become satisfied/unsatisfied during the random walk, shows that on the formulas with medium and long range connections the RW procedure converges much more quickly than one would expect.

---

[4] One could argue that, in practice, a local search algorithm would never encounter such a formula, since one generally removes unit clauses before running a local search procedure. However, note that the forced setting of x_1 and x_2 to True could be a hidden consequence of another set of non-unit clauses.

Let $f(s)$ denote the random variable that represents the number of flips RW takes before reaching a satisfying assignment when starting from an initial (arbitrary) assignment s. We want to determine the expected value of $f(s)$, *i.e.*, $\mathbf{E}(f(s))$, given any initial starting assignment s.

The following theorems summarize our main formal results.

Theorem 2. *Given a ternary chain formula $F_{3chain,i-2}$, starting at a random assignment s, the expected value of $f(s)$, the number of flips to reach a satisfying truth assignment, scales exponentially in n.*

Theorem 3. *Given a ternary chain formula $F_{3chain,low(i)}$ and let s be an arbitrary truth assignment, we have for the expected number of flips for the RW procedure:*

a) $\mathbf{E}(f(s)) = O(n \cdot n^{\log n})$, *for low(i)*$= \lfloor \frac{i}{2} \rfloor$

b) $\mathbf{E}(f(s)) = O(n^2 \cdot (\log n)^2)$, *for low(i)*$= \lfloor \log i \rfloor$.

The proofs of these theorems require the solution of a series of recurrence relations that characterize the expected time of the random walk process. As noted above, instead of analyzing a single random walk, the random walk process is decomposed into separate subsequences depending on the clauses and variables involved. The proofs are rather involved and are omitted here because of space limitations. The interested reader is referred to the long version of the paper, see `www.cs.cornell.edu/home/selman/weiwei.pdf`.

Theorems 2 and 3 provide further evidence that links capturing longer range dependencies among variables can dramatically accelerate the convergence of RW procedures, just as we saw for biased RW on the binary chain formulas. The left panel in Figure 7 empirically illustrates our scaling results. Note the logarithmic vertical scale. We see clear exponential scaling for the chain formulas with only short range connections. The other graphs curve down, indicating better than exponential scaling, with the best scaling for the long range dependency links (polynomial). Finally, in the right panel of the figure, we consider the effect of adding a greedy bias in the RW. We consider the formulas with medium range dependency links. We see some improvement for RWF and WalkSat over RW, but no dramatic difference in scaling. So, the range of dependency links (short, medium, or long) in the 3chain formulas is the main factor in determining the scaling both for both biased and unbiased random walks.

4   Empirical Results on Practical Problem Instances

We have shown both theoretically and empirically that the performance of the WalkSat procedure improves when long range connections are added to our chain formulas. Given that chains of dependent variables commonly occur in practical problem instances, we hypothesize that similar performance improvements can be obtained on real-world formulas. In this section, we test this hypothesis on a suite of practical problem instances.

In order to apply our technique to a real-world formula, however, we need to be able to discover the underlying variable dependencies and add long-distance links to the corresponding implication graph. Quite fortunately, Brafman's recent 2-Simplify method [5] looks exactly for such long range dependencies. The procedure improves the performance of SAT solvers by simplifying propositional

Table 1. Number of instances WalkSat (noise = 50) solved (averaged over 10 runs) on Velev benchmark [28, 29] (100 instances total). Prepocessing using 2-Simplify; α is the redundancy level.

Formulas	< 40 sec	< 400 sec	< 4000 sec
$\alpha = 0.0$	15	26	42
$\alpha = 0.2$	**85**	**98**	**100**
$\alpha = 1.0$	13	33	64

formulas. More specifically, 2-Simplify constructs a binary implication graph and adds new clauses (links) that are based on the transitive closure of existing dependencies. (In current work, we are extending this process by adding links implied by ternary clause chains.)

We built our preprocessor upon Brafman's implementation. More precisely, his code simplifies a formula in the following steps. First, it constructs an implication graph from binary clauses, and collapses strongly connected components that exist in this graph. Second, it generates the transitive closure from existing links in the graph, which enables the algorithm to deduce some literals through binary resolution and hyper-resolution, and remove the assigned variables from the graph. Third, it removes all transitive redundant links to keep the number of edges in the graph minimal. Finally, the graph is translated back into binary clauses. Note that Brafman's procedure eliminates all the transitively redundant links in the third step. The procedure was designed mainly for speeding up DPLL style procedures, which can easily recover the transitive closure dependencies via unit propagation. However, based on our earlier results on the chain formulas, to speed up local search methods, we need to keep exactly those implied clauses that capture long range dependencies. That is, WalkSat may actually exhibit better performance when working on formulas with a fraction of redundant long range links kept in place. To implement this strategy, we modified the 2-Simplify procedure to remove implied transitive links from the transitive closure graph in a probabilistic manner. More specifically, we introduce a *redundancy parameter*, α, which gives the probability that we keep a redundant link. So, in the third step of the 2-Simplify procedure, we step through the redundant links and removes each one with probability $1 - \alpha$.

We will now show that this rather straightforward approach makes a tremendous difference in WalkSat's ability to find a satisfying truth assignment on structured benchmark problems. (Of course, given our earlier results on the chain formulas, this is not completely unexpected.) We ran WalkSat on the SAT-1.0 benchmarks used recently by Velev [28, 29] to evaluate a collection of 28 state-of-the-art SAT solvers. The suite consists of 100 formulas encoding verification problems for superscalar microprocessors. The instances have been shown to be challenging for both local search methods and DPLL procedures [29]. Moreover, they represent an important practical application. From Velev's results and to the best of our knowledge, no local search methods is able to solve more than 65 instances in under 40 seconds of CPU time per instance.

Our experiments were performed on a 600 MHz Pentium II processor running Linux. See Table 4 for our results. The first row shows the results for the original formulas in SSS-SAT-1.0 suite after simplification using Brafman's preprocessor ($\alpha = 0.0$). The second row shows the results when a fraction of redundant clauses is added ($\alpha = 0.2$). (The preprocessing time is included in the total run time.)

It is clear from the table that WalkSat's performance improves dramatically when adding a fraction of redundant clauses: solving 85 instances within 40 sec-
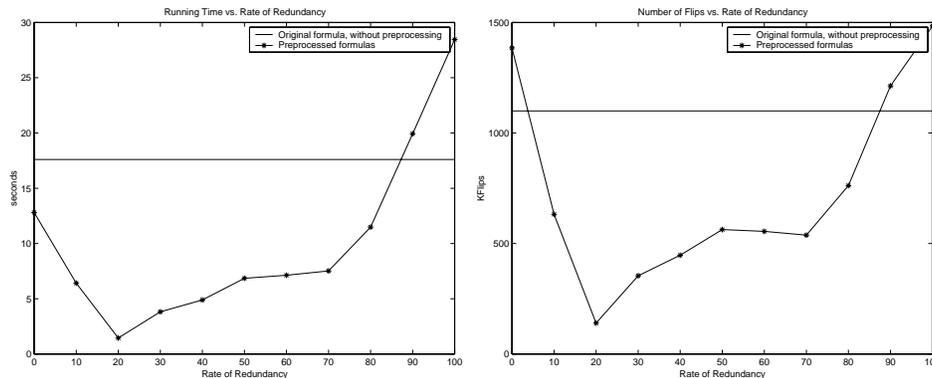
Fig. 8. Different levels of redundancy on instance `dlx2_cc_bug01.cnf` from SAT-1.0 suite [28]. Results are averaged over 100 runs. Noise level WalkSAT set at 50.

onds with 20% redundancy, compared to just 15 with no redundancy; moreover, 98 instances can now be solved in under 400 seconds, compared to 26 without redundancies. In future work, we will provide a comparison with the performance on the original formulas without any preprocessing. Brafman's current simplifier, unfortunately, leads to theories that are logically slightly weaker than the original theory. This makes a comparison with the performance on the original formulas difficult. Nevertheless, our experiments in Table 4 clearly demonstrate the benefit of adding long range dependencies.

We observed a clear optimal value for the rate of redundancy α. In particular, adding all implied transitivity constraints can be harmful. This is apparent from the third row in Table 4. With $\alpha = 1.0$, we add all redundant transitivity constraints uncovered by the simplifier. Clearly, adding only a randomly selected fraction of implied transitivity constraints (20% in this case) gives us much better performance. Figure 8 shows the number of flips until solution and the run time of the WalkSat algorithm for different α values. These results are consistent with our observations for adding random long range links to our binary chain formula. In that setting, there was also a clear optimal level of redundancy for the biased random walk approach. See Figure 5.

Cha and Iwama [6] also studied the effect of adding clauses during the local search process. They focus on clauses that are resolvents of the clauses unsatisfied at local minima, and their randomly selected neighbors. This meant that the added clauses captured mainly short range information. Our results suggest that long range dependencies may be more important to uncover.

In summary, our experiments show that the insights obtained from our study of the chain formulas can be used to speed up local search methods on SAT problems from practical applications. In particular, we found a clear practical benefit of adding implied clauses that capture certain long range variable dependencies and structure in the formulas. Interestingly, to make this approach work one has to tune the level of redundancy added to the formulas. Somewhat surprisingly, the total number of added clauses was only about 4% of the original number of clauses.

5 Conclusions

We have provided theoretical and empirical results showing how one can speed up random walk style local search methods. The key idea is to introduce constraints that capture long range dependencies. Our formal analysis for unbiased random walks on ternary chain formulas shows how the performance of RW varies from exponential to polynomial depending on the range of the dependency links, with long range links leading to a tractable sub-case. We believe this is the first tractable 3CNF class identified for unbiased random walks. On the binary chain formulas, we saw how such constraints can also improve the performance of biased random walks. On the binary chains with added implied clauses, biased random walks become almost as effective as unit propagation used in DPLL procedures. Finally, we showed how on practical instances based on hardware verification problems adding a fraction of long range dependencies significantly improves the performance of WalkSat on such formulas.

Our results so far show the promise of our technique for speeding up local search methods on structured benchmarks domains. However, we believe that there is still much room for further improvements. In particular, it should be possible to develop formula preprocessors that uncover other types of dependencies between variables that may be useful for further accelerating random walks. Again, a methodology guided by an analysis of formulas where biased and unbiased random walks exhibit extreme behavior should be helpful.

Acknowledgements

References

1. F. Bacchus and P. van Beek. On the Conversion between Non-Binary and Binary Constraint Satisfaction Problems. In *Proceedings of AAAI-98*, pages 311-318, 1998.
2. M. Davis and H. Putnam. A Computing Procedure for Quantification Theory. *Journal of the ACM*, 7: 201-205, 1960.
3. M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem-Proving. *Communications of the ACM*, 5:394-397, 1962.
4. B. Bonet and H. Geffner. Planning as Heuristic Search. *Artificial Intelligence, Special issue on Heuristic Search*. Vol 129 (1-2) 2001.
5. R. I. Brafman. A Simplifier for Propositional Formulas with Many Binary Clauses, In *Proceedings IJCAI-2001*, Seattle, WA, pages 515-520, 2001.
6. B. Cha and K. Iwama. Adding New Clauses for Faster Local Search. In *Proceedings AAAI-96*, pages 332-337, 1996.
7. L. Drake, A. Frisch, and T. Walsh. Adding resolution to the DPLL procedure for Satisfiability. In *Proceedings of SAT2002*, pages 122-129, 2002.
8. I. Gent and T. Walsh. An Empirical Analysis of Search in GSAT. *JAIR*, 1993.
9. C. P. Gomes, B. Selman, N. Crator, and H. Kautz. Heavy-Tailed Phenomenia in Satisfiability and Constraint Satisfaction Problems. *Journal of Automated Reasoning*, Vol 24, No 1-2, February 2000.
10. J. Gu. Efficient Local Search for Very Large-Scale Satisfiability Problems. *SIGART Bulletin* 3(1): 8-12, 1992.

11. E. A. Hirsch and A. Kojevnikov. Solving Boolean Satisfiability Using Local Search Guided by Unit Clause Elimination. In Proc. of CP-2001, 2001.
12. H. Kautz and B. Selman. Unifying SAT-based and Graph-based Planning. In *Proceedings of IJCAI-99*, Stockholm, 1999.
13. J. Kleinberg. Navigation in a small-world. *Nature*, **406**, 2000.
14. S. Lin and B. W. Kernighan. An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research*, Vol. 21, 2, pages 498–516, 1973.
15. D. A. McAllester, B. Selman, and H. A. Kautz. Evidence for Invariants in Local Search. In *Proceedings of AAAI-97*, pages 321-326, 1997.
16. P. Morris. Breakout method for escaping from local minima. *Proc. AAAI-93*, Washington, DC (1993).
17. S. Minton, M. D. Johnson, A. B. Philips, and P. Laird. Solving Large-Scale Constraint Satisfaction and Scheduling Problems Using a Heuristic Repair Methods. *Artificial Intelligence* 58, pages 161-205, 1990.
18. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Engineering a Highly Efficient SAT Solver. *38th Design Autom. Conference (DAC 01)*, 2001.
19. C.H. Papadimitriou. On Selecting a Satisfying Truth Assignment. In *Proceedings of the Conference on the Foundations of Computer Science*, pages 163-169, 1991.
20. A. J. Parkes and J. P. Walser. Tuning Local Search for Satisfiability Testing. In *Proc. AAAI-96*, 1996.
21. S. Prestwich. SAT Problems with Chains of Dependent Variables. Unpublished technical note, 2001.
22. U. Schoening. A Probabilistic Algorithm for k-SAT and Constraint Satisfaction Problems. In *Proceedings of FOCS*, 1999.
23. D. Schuurmans, F. Southey, and R. C. Holte. The Exponential Subgradient Algorithm for Heuristic Boolean Programming. In *Proc. IJCAI-2001*, 2001.
24. B. Selman, H. Kautz, and B. Cohen. Local Search Strategies for Satisfiability Testing. *2nd DIMACS Challenge on Cliques, Coloring and Satisfiability*, 1994.
25. B. Selman, H. Kautz, and D. McAllester. Ten Challenges in Propositional Reasoning and Search. In *Proceedings IJCAI-97*, 1997.
26. B. Selman, H. J. Levesque, and D. G. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proceedings AAAI-92*, pages 440-446, 1992.
27. T. Suyama, M. Yokoo, and A. Nagoya. Solving Satisfiability Problems on FPGAs Using Experimental Unit Propagation. In *Proceedings of CP-99*, 1999.
28. M. N. Velev. Benchmark suites SSS-SAT-1.0, SAT-1.0, October 2000. http://www.ece.cmu.edu/∼mvelev
29. M. N. Velev, and R. E. Bryant. Effective Use of Boolean Satisfiability Procedures in the Formal Verification of Superscalar and VLIW Microprocessors. *38th Design Automation Conference (DAC '01)*, June 2001, pp. 226-231.
30. J.P. Walser, R. Iyer and N. Venkatasubramanyan.An Integer Local Search Method with Application to Capacitated Production Planning. In *Proceedings of AAAI-98*, Madison, WI, 1998.
31. T. Walsh. Search in a Small World. In *Proceedings of IJCAI-99*, 1999.
32. D.J. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, **393**, 440-442 (1998).
33. Z. Wu and B. W. Wah. An Efficient Global-Search Strategy in Discrete Lagrangian Methods for Solving Hard Satisfiability Problems. *Proc. AAAI-2000*, 2000.
34. H. Zhang. SATO: An Efficient Propositional Prover. *International Conference on Automated Deduction (CADE 97)*, LNAI 1249, Springer-Verlag, 1997.
35. Available at http://www.cs.washington.edu/homes/kautz/walksat/