A local search algorithm for 3SAT

Uriel Feige and Dan Vilenchik

July 7, 2004

Abstract

We present a fairly simple and natural algorithm for 3SAT, which can be viewed as a particular instantiation of the well known k-opt heuristic. We prove that for most formulas drawn from certain random and semi-random distributions on satisfiable formulas, the algorithm finds a satisfying assignment in polynomial time.

1 Introduction

For the well known problem of 3SAT, the input is a 3CNF formula on n Boolean variables. Each of the clauses of a 3CNF formula is the disjunction (logical *or*) on three literals, where a literal is either a variable or its negation. A 0/1 assignment to the variables (and the complement to their negation) is a *satisfying assignment* if in every clause at least one literal is assigned the value 1. The goal is to find a satisfying assignment, if one exists. The problem 3SAT is well known to be NP-hard.

In this paper we shall consider heuristics for 3SAT. By a heuristic we mean an efficient (polynomial time) algorithm that is not guaranteed to produce the correct result, but in some interesting cases it does. Factors of importance in the design of heuristics include the resources that it consumes (in terms of running time and space of the algorithm, and in terms of programming effort needed in order to implement it), and its quality in terms of how likely it is to actually help and produce good results. We are not aware of universally agreed upon criterions for evaluating the quality of heuristics. Evidence that is typically given to show that a certain heuristic is good often includes reports of experimental results on benchmark instances, reports of results achieved on inputs encountered in practice, and rigorous or semi-rigorous analysis showing that if inputs are drawn from certain distributions then the heuristic is likely to solve them.

In this work, we consider a version of the well known k-opt heuristic and adapt it to 3SAT. We rigorously prove that the resulting algorithm almost always finds a satisfying assignment for inputs drawn from some natural distribution on satisfiable 3CNF formulas. We make no claims regarding the perfomance of the algorithm on instances that one encounters in practice, or on well known benchmarks for 3SAT. In fact, we suspect that our algorithm is inferior to some other algorithms that were developed for 3SAT. The main goals of the current work are twofold. One is to show that a fairly natural algorithm for 3SAT can be rigorously analysed under some natural distribution, and proved to produce good results. The other is to highlight the use of semirandom models (as explained in Section 2) as a driving force in the design and analysis of heuristics.

There is a huge body of work related to our current work. We mention some of it, selectively. The k-opt heuristic is well known, see [16] for example. Comparative results for various heuristics for 3SAT can be found in [8]. Analysis of an algorithm that works in a probabilistic model similar to the one that we consider can be found in [11] (but see Section 2 for more information). Rigorous analysis of the performance of various heuristics on very sparse random 3CNF formulas appears in [1, 14, 2], among others. Heuristics for refuting 3CNF formulas (showing that no satisfying assignments exist) are considered in [6, 4, 12], among others.

This work is based on the MSc thesis of the second author, though the presentation here differs from the presentation in the thesis. More detailed proofs for some of the technical claims can be found in [17].

2 Planted assignment models

We consider the *planted assignment model* for generating random satisfiable 3CNF formulas. The model is parameterized by n (the number of variables) and d (the *density*). One first selects a random assignment ϕ to the variables. Then one generates $m = d \cdot n$ clauses independently at random. Every clause is generated by picking uniformly at random a set of three variables, and then picking the polarity of the variables at random, subject to the constraint that the clause is satisfied by ϕ. (Out of the $2^3 = 8$ ways of picking the polarity, seven result in a clause satisfied by ϕ, and one of these seven possibilities is chosen at random.)

Typical formulas generated in the planted assignment model contain "statistical hints" pointing to the assignment ϕ. For a variable x_i, one expects a fraction of $4/7 > 1/2$ of its ocurrences to be of the same polarity as its polarity in ϕ. Hence assigning a variable in agreement with the polarity of most of its ocurrences is likely to also be in agreement with ϕ. It is not hard to show that the fraction of variables assigned incorrectly by this majority vote procedure is almost surely exponentially small in the density d of the formula. (For more details, see Section A.1.) In particular, when $d > c \log n$ for a sufficiently large constant c, the majority vote recovers the original ϕ.

We shall be interested in the case that d is a "large" constant but independent of n. In this case, most variables are set in agreement with ϕ by the majority vote, but a small fraction of the variables are set with flipped polarity. When n is large enough, the majority vote assignment is unlikely to satisfy the input formula. The challenge is to find a satisfying assignment (not necessarily the original ϕ).

Flaxman [11] (following an algorithmic pattern originating in [3, 7]) describes a polyomial time algorithm that almost surely (over the choice of input formula) finds a satisfying assignment in the planted assignment model. The algorithm that we present in Section 3 works as well. Both algorithms start with the majority vote assignment. (Flaxman considers a more general distribution on formulas with planted assignments, in the sense that not all seven options for the polarities within a clause are equally likely. For the more general distribution Flaxman gets an initial assignment based on statistics over pairs of variables, using spectral techniques, rather than statistics over individual variables, using the majority vote. However, for the distribution considered in the current paper the majority vote assignment can safely replace the spectral based assignment in Flaxman's algorithm, as Flaxman himself points out.) The difference between our algorithm and that of Flaxman is in how one modifies the majority vote assignment so as to get a satisfying assignment. We use an algorithm based on k-opt, to be described in Section 3. Flaxman uses a multi-phase procedure. He first unassigns some of the variables that exhibit untypical statistical properties in terms of the number and type of clauses in which they appear (exact definitions appear in [11]). The rest of the variables are assumed (and Flaxman justifies this assumption by analysis) to have values that agree with ϕ. The input formula is then simplified by substituting in the values of these variables, and removing satisfied clauses. Then a residual formula remains on the set of unassigned variables. Flaxman shows that the residual formula is most likely composed of several variable-disjoint subformulas, and that each such subformula contains at most $\log n$ variables. Then a satisfying assignment is found using exhaustive search over assignments to each subformula separately.

Let us now explain how and why we arrived at our algorithm. The motivation was the view that algorithms that are designed for random models are often overfitted to the particular model, and are not robust to slight changes in the model. To test whether a particular algorithm is robust, it is suggested to replace the random model by a semirandom model. (Semirandom models were first introduced in [5]. The presentation here follows the principles outlined in [9].) In the semirandom model, an adversary can change the random input in a way that may superficially appear to be harmless. If the algorithm fails to work in the semirandom model, this is an indication that it is not robust. In this case one tries to modify the algorithm (or come up with a completely different algorithm) so that it does work. Hopefully, by this one arrives at an algorithm that is not only more robust, but also more natural. Examples of this approach appear in [9, 10, 15].

The semirandom model that we consider is the following. First a formula is generated at random from the planted assignment model. Then an adversary is allowed to add arbitrary clauses to the formula, provided that all three variables in these clauses have exactly the same polarity that they have in ϕ. The order of the clauses is shuffled at random so as to hide the information of which clauses are the random clauses and which are the adversarial ones.

Intuitively, the addition of adversarial clauses should only help in finding a satisfying assignment, as all three variables in them correctly hint to their polarity in ϕ. In particular, the majority vote assignment can only benefit from the adversarial clauses. Nevertheless, Flaxman's algorithm fails in this semirandom model. A minor problem that the adversary can create for Flaxman's algorithm is to fool the unassignment phase by adding adversarial clauses that make the statistics of many variables untypical (according to Flaxman's original definition). This problem can be overcome with a more careful definition of what it means for a variable to be typical. (Let us note here that this modification to Flaxman's algorithm already achieves part of the goals of introducing the semi-random model, because it forces one to think more carefully on which statistical properties of the variables are the relevant ones for the algorithm to work, and are not sensitive to the addition of the adversarial clauses.) However, a more serious problem is that when adversarial clauses are added, the residual formula need not decompose into small disjoint subformulas, making the exhaustive search approach inapplicable.

Our algorithm was designed so as to work also in the semirandom model described above (and indeed it works). Hence, in some exact sense, our algorithm is more robust than Flaxman's algorithm. Moreover, it is our view (but this is of course debateable) that our algorithm is more natural than Flaxman's algorithm, and that its analysis is simpler (even though the underlying principles for the analysis are similar).

3 The algorithm

Our algorithm is based on a version of the k-opt heuristic. Let us first explain how one would apply the k-opt heuristic for solving *max-3SAT*. The input to the max-3SAT problem is a 3CNF formula with n variables and m clauses. The *value* of a feasible solution (namely, an assignment to the variables) is the number of clauses that are satisfied by it. The goal is to find a solution of maximum value.

To apply the k-opt heuristic, one can imagine all feasible solutions as vertices of a graph, where two vertices are joined by an edge if their corresponding assignments agree on all variables but one. We call this graph the *Hamming graph*. The value of a vertex in the graph is the value of the corresponding solution. The algorithm starts at an arbitrary vertex on this graph. Then, in every step of the algorithm, it seeks to move to a vertex of higher value, and of distance no more than k from the current vertex. Such a move if performed is called an *improving move*. If more than one improving move is possible in a certain step, then there is some arbitration mechanism that chooses among the possible moves. When no improving move exists, the algorithm is said to have reached a *local optimum*. The algorithm stops and outputs the solution corresponding to the local optimum.

The above description left some aspects of the algorithm unspecified, such as which starting vertex to choose, which arbitration rule is used in case of more than one possible move, and which values of k give a favorable tradeoff between running time and quality of solution. These aspects are important and have influence on the quality of solutions found and on the running time of the algorithm. They will be addressed in the context of our algorithm for 3SAT.

Let us now present our k-opt for 3SAT. We shall use the same Hamming graph as the one used for max-3SAT. But for the issue of the values given to vertices, or what we call the *landscape*, we shall make a perhaps nonstandard choice. Strictly speaking, in the 3SAT problem all nonsatisfying assignments are equally bad as solutions, unlike the case in max-3SAT. Hence a naive approach may give every satisfying assignment the value 1, and every nonsatisfying assignment the value 0. However, this gives a landscape that is too “flat” for the k-opt heuristic to be of any value. To get greater variablity in the landscape, hopefully creating a drift towards satisfying assignments, one may associate with each vertex of the search graph the same value that it would get in the max-3SAT problem, namely, the number of clauses that it satisfies. We call this the max-3SAT landscape.

In the current paper, we use a different landscape. We call our landscape the *set-based* landscape. We give every vertex a value that is a subset in $\{1, \ldots, m\}$, rather than an integer in that range. Specifically, the value of a vertex is the set of clauses that it satisfies, rather than the number of clauses that it satisfies. This induces a partial order among vertices, based on set containment. A vertex has better value than another vertex if the set of clauses satisfied by the former vertex strictly contains the set of clauses satisfied by the latter vertex.

Let us point out a useful property of the set-based landscape in the context of k-opt. Observe that the Hamming graph has degree n, and finding an improving move might involve examining $\sum_{i=1}^{k} \binom{n}{i}$ vertices of the graph. This cost may be prohibitingly large even for relatively small values of k. However, for the set-based landscape we can do better.

Proposition 1 *For the set-based landscape, it suffices to inspect $2^k n$ vertices in order to find an improving move (if one exists).*

Proof: Given an assignment ϕ to the variables, consider the following search tree of depth k. The root of the tree has degree at most n, and every internal vertex has degree at most 2. Hence the total number of vertices in the search tree is at most $2^k n$. Every edge of the search tree is labeled by a variable (according to some rule that will be described shortly). No variable appears twice on the same root-to-leaf path, or on two edges that are incident with the same vertex. With every vertex v of the tree we associate an assignment ϕ_v which is identical to the assignment ϕ, except that all variables on the path from the root to v are flipped. The edges leading out of the root are labeled by the variables that appear in clauses that are not satisfied by ϕ. The edges leading out of a vertex v are labeled as follows. If ϕ_v satisfies all clauses satisfied by ϕ, then we make v a leaf of the search tree. If there are clauses satisfied by ϕ but not by ϕ_v, we pick one of them arbitrarily (denote it by C_v), and label the edges leading out of v by those variables from C_v that are not already flipped in ϕ_v. Note that there are at most two such variables, because the variable of ϕ that satisfied C_v is necessarily already flipped in ϕ_v. This completes the description of the search tree. The algorithm inspects all assignments ϕ_v in the search tree (say, in depth first search order), and if one of them is an improving assignment, this assignment is output. Clearly, the distance of this assignment from ϕ is at most k.

It remains to show that if there is an improving assignment ϕ' at Hamming distance at most k from ϕ, then at least one of the ϕ_v appearing in the search tree is an improving assignment (though not necessarily ϕ' itself). The point is that there is a path starting in the root that at every move decreases the Hamming distance to ϕ'. We call such a path a *correcting path*. As the depth of the search tree is k, then either ϕ' is reached along this path, or an improving assignment is found earlier. We now describe a correcting path (there may be several such paths). There is some clause C that is satisfied by ϕ' but not by ϕ. From the root, move along an edge labeled by a variable that satisfies C in ϕ'. For an internal vertex v of the tree reached by the correcting path, there is some clause C_v that is satisfied by ϕ but not by ϕ_v, and is used in order to label the edges leading out of v. This clause must also be satisfied by ϕ'. Follow an edge labeled by the variable that satisfies C in ϕ'. (Note that this cannot be a variable that was already flipped, because all flipped variables along the correcting path necessarily agree with ϕ'.) □

Hence for the set-based landscape, we can run k-opt in polynomial time as long as $k = O(\log n)$.

The k-opt algorithm with the set-based landscape may hit a local (rather than global) optimum rather quickly. In fact, for the same value of k, every local optimum of the max-3SAT landscape is a local optimum for the set-based landscape, but the converse in general does not hold. A careful choice of the starting vertex may in some cases help the set-based algorithm avoid hitting a local optimum. In the context of the random formulas that we shall consider, the *majority vote* heuristic offers a good starting point. This heuritic assigns a variable in agreement with the majority of its occurances as a literal. Namely, if a variable appears more times positively than negated, it is assigned the value true, and false otherwise. (Ties can be broken either at random, or arbitrarily, or by some other heuristic rule. One can ensure that at least half the tied variables are set correctly by considering two different starting vertices, one with all tied variables assigned positively, and the other with all tied variables assigned negatively.)

We can now give a complete description of our algorithm. To make it an algorithm that never fails, we do not work with a fixed value of k, but rather with the smallest value of k that makes it succeed.

1. Produce an initial assignment using the majority vote.
2. Initialize $k = 1$.
3. Apply the k-opt heuristic with the set-based landscape on the Hamming graph until a local optimum is reached.
4. If a satisfying assignment is reached, output this assignment.
5. Else, if $k = n$, output "the formula is not satisfiable".
6. Else, increase the value of k by 1 and return to step 3.

To implement one step of the k-opt heuristic we recommend to use depth first search on the search tree described in Proposition 1. We note that the degree of the root of the search tree decreases in every step, because for every variable that is ever flipped by our k-opt algorithm, all its clauses remain satisfied for the remaining run of the algorithm. (For example, if it was flipped from positive to negative, then prior to the flip all clauses in which is appears positively were satisfied, and hence are required to remain satisfied. All clauses in which is appears negatively are satisfied after the flip, and hence they too are required to remain satisfied.) Hence the total number of improving k-moves performed by the algorithm is at most n. The running time of the algorithm is polynomial in n and m, and exponential in the largest value of k reached by the algorithm. The space used by the algorithm is linear in m and n, regardless of the value of k that is reached.

4 On cores and propagation graphs

Here we present general principles for analysing our k-opt algorithm for 3SAT, for a fixed value of k. We assume here that the input formula F has some satisfying assignment ϕ, and the k-opt algorithm has an initial assignment ψ. We shall present sufficient conditions for k-opt to converge to a satisfying assignment (though not necessarily to ϕ). The running time of the algorithm is then at most $2^k\text{poly}(n)$, which is polynomial in n whenever $k = O(\log n)$.

Moves of the k-opt heuristic flip variables. The choice of which variables are actually flipped is not fully specified in our algorithm, as this requires an arbitration rule when there are several possible improving moves, and we did not specify any particular arbitration rule. However, given an initial assignment ψ, there are certain variables that will never be flipped by k-opt, regardless of the arbitration rule that is used. We say that these variables are *fixed* by ψ, and the other variables are called the *nonfixed* variables. As a simple example (for simplicity, we consider here 2SAT rather than 3SAT), assume that two of the clauses of the input formula are $(x_1 \vee x_2)$ and $(x_1 \vee \bar{x_2})$. Then if $x_1 = 1$ in ψ, the two clauses are satisfied, and flipping the value of x_1 makes one of them not satisfied, regardless of the value of x_2. Hence in our set-based landscape for k-opt, there is no improving move that involves flipping x_1, regardless of the value of k. In this specific case, not flipping x_1 appears to be a desirable feature of the algorithm, because $x_1 = 1$ in every satisfying assignment for F. However, in other cases, the existence of fixed variables may prevent k-opt from reaching a satisfying assignment.

Given the set of variables fixed by ψ and their values in ψ, one can substitute these values in F and simplify the formula, by removing clauses that are satisfied by the fixed variables, and removing occurences of the fixed variables from the remaining clauses. We call this subformula (involving only nonfixed variables) the *residual* formula. Possibly, this formula is not satisfiable, either because it already contains an empty clause, or more generally because every assignment to the nonfixed variables fails to satisfy it.

Proposition 2 *A necessary condition for k-opt to find a satisfying assignment starting with initial assignment ψ is that the residual formula (as defined above) is satisfiable.*

The converse of Proposition 2 is not necessarily true. The residual formula may be satisfiable, and still the k-opt algorithm may fail to find a satisfying assignment. For example, consider a residual formula composed of the three clauses (x_1), $(x_1 \vee x_2)$ and $(\bar{x_1} \vee \bar{x_2})$, and assume that in ψ, $x_1 = x_2 = 0$ (hence only the third clause is satisfied), and that we are running the 1-opt algorithm. The residual formula has a satisfying assignment, namely $x_1 = 1$ and $x_2 = 0$. Moreover, this assignment is reachable by one flip from ψ. However, 1-opt may choose at its first step to flip x_2 (satisfying the last two clauses), and then it is stuck in a local optimum (every single flip makes one of the two last clauses not satisfied). Perhaps a more striking example is the following. The residual formula contains the six clauses (x_1), (x_2), (x_3), $(x_1, \bar{x_2}, \bar{x_3})$, $(\bar{x_1}, x_2, \bar{x_3})$ and $(\bar{x_1}, \bar{x_2}, x_3)$, and ψ has $x_1 = x_2 = x_3 = 0$ (whereas $x_1 = x_2 = x_3 = 1$ is the only satisfying assignment). For 1-opt, every variable can be flipped on the first move (and hence indeed is nonfixed), but then a local optimum is reached and the 1-opt algorithm fails to converge to a satisfying assignment.

We now give a sufficient condition for k-opt to find a satisfying assignment, regardless of the arbitrartion rule used. This condition involves the construction of a certain directed graph, that we call the *propagation graph.* The graph depends on the initial assignment ψ, and also on a satisfying assignment ϕ. Namely, for the same ψ there may be several different propagation graphs, depending on the choice of satisfying assignment ϕ. We let $G_{\psi,\phi}$ denote the propogation graph constructed from (ψ, ϕ). The vertices of $G_{\psi,\phi}$ are the nonfixed variables with respect to ψ. There is a directed edge from vertex x_i to vertex x_j if there is a clause C in the residual formula with the following properties:

1. x_i appears in C in a polarity that disagrees with its polarity in the satisfying assignment ϕ.

2. x_j appears in C in a polarity that agrees with its polarity in ϕ.

For clarity, we state in more explicit terms the condition that C is a clause of the residual formula. For C in F (with x_i and x_j nonfixed), this means that the third variable in C is either a nonfixed variable, or a fixed variable with polarity that disagrees with its polarity in the initial assignment ψ.

This completes the description of the propagation graph. For vertex x_i in $G_{\psi,\phi}$, let $P(x_i)$ be its *propagation set*, the set of vertices reachable from x_i via a directed path in the propagation graph.

Theorem 3 *Given a formula F and an initial assignment ψ, the algorithm k-opt always finds a satisfying assignment (regardless of the arbitrartion rule) if F has some satisfying assignment ϕ with respect to which both the following conditions hold:*

1. *For every variable that is fixed with respect to ψ, its assignment in ψ is identical to its assignment in ϕ.*

2. *For every vertex x_i in the propagation graph $G_{\psi,\phi}$, its propagation set satisfies $|P(x_i)| \leq k-1$.*

Proof: Let ψ' be an arbitrary assignment reached after an arbitrary number of moves of k-opt, starting at ψ, and assume that ψ' is not a satisfying assignment. We show that an improving move exists at ψ'. Consider an arbitrary clause C not satisfied by ψ'. As this clause is satisfied by ϕ, there must be a variable (say, x_i) used by ϕ to satisfy C. Consider the assignment ψ'' obtained by flipping x_i in ψ' so that it now agrees with its assignment in ϕ. The clause C is satisfied by ψ'', but some other clauses that were satisfied by ψ' might become unsatisfied due to the flipping of x_i. Let C' be such a clause. Then x_i necessarily appears in C' with a polarity that disagrees with its polarity in ϕ. Moreover, ϕ satisfies C', so C' must contain a variable x_j with polarity that agrees with its polarity in ϕ. This variable cannot be one of the fixed variables with respect to ψ, because ϕ, ψ, ψ' and ψ'' all agree on the values of the variables fixed with respect to ψ, and we assumed that ψ'' does not satisfy C'. This implies that (x_i, x_j) is an edge of the propagation graph $G_{\psi,\phi}$. The k-opt algorithm may choose to flip also x_j, and then C' becomes satisfied, and other clauses may become unsatisfied. The main point to notice is that all variable that are considered for flipping are reachable from x_i in the propagation graph. Hence if $k-1 \geq |P(x_i)|$, there is an improving k-opt move. □

Remark: The reader may wonder whether Theorem 3 can be strengthened as follows. Rather than require that $|P(x_i)| < k$ for every vertex, require only that this holds for at least one vertex, and similarly for every vertex induced subgraph of $G_{\psi,\phi}$. In the special case of 1-opt, this condition is equivalent to requiring that $G_{\psi,\phi}$ does not have any directed cycle. However, this does not quite work for our algorithm. Consider for example the residual formula with the two clauses (x_1) and $(\bar{x_1}, x_2)$, with $x_1 = x_2 = 0$ in ψ. The propagation graph has a single edge (x_1, x_2) and hence has no cycles, but 1-opt makes no progress. It can flip x_2, but will not do that because this is not an improving move, but rather a *neutral* move that does not change the set of satisfied clauses. We may augment the k-opt heuristic by allowing also neutral moves, but this has to be done with care as it introduces the possibility of cycling.

To construct the graph $G_{\psi,\phi}$ one needs to know which are the fixed variables with respect to ψ. It is often easier to base the analysis of k-opt only on a subset of the fixed variables, rather than on all of them. A convenient subset of this form is what we call a *core*.

Definition 1 *A set S of variables are a k-*core *for formula F with respect to assignment ψ if the following conditions hold:*

1. *The subformula F_S induced on S (namely, the set of all clauses of F for which all three variables belong to S) is satisfied by ψ.*

2. *Every other assignment for the varaibles of S that satisfies F_S differs from ψ in the values that it assigns to strictly more than k variables in S.*

Clearly, k-opt cannot flip the value of a k-core variable, as this forces the flip of a total of more than k variables in order to get an improving move. (Otherwise, the respective F_S is not satisfied, whereas it is satisfied by ψ.)

There may be many different cores with respect to ψ. If the residual formula with respect to ψ is satisfiable, then the union of cores is also a core, and there is a unique maximum core. In general, we shall be interested in finding some "sufficiently large" core, rather than the maximum core.

Given a formula F, an assignment ψ and a core S we may consider the residual formula that remains after substituting in F the values of ψ for the core variables. Given also a satisfying assignment ϕ, we may construct a propagation graph $G_{S,\phi}$ whose vertices are the noncore variables, and edges are defined analogously to the case of $G_{\psi,\phi}$. The proof of Theorem 3 then extends to prove the following theorem.

Theorem 4 *Given a formula F, an initial assignment ψ and a k-core S with respect to ψ, the algorithm k-opt always finds a satisfying assignment (regardless of the arbitrartion rule) if F has some satisfying assignment ϕ with respect to which both the following conditions hold:*

1. *For every variable in S, its assignment in ψ is identical to its assignment in ϕ.*

2. *For every vertex x_i in the propagation graph $G_{S,\phi}$, its propagation set satisfies $|P(x_i)| \leq k - 1$.*

5 Performance in planted assignment model

We show that the k-opt algorithm finds a satisfying assignment in polynomial time with high probability over the choice of input formula F, where the input formula is chosen at random from the planted assignment model, and when d (the density) is sufficiently large. (We need $d > d_0$ where d_0 is some constant that we do not attempt to compute here.) We then show that the proof extends in a straightforward way to the semirandom model. Our proof involves an application of Theorem 4.

For the input formula F with a planted assignment, we use ϕ to denote the planted assignment, and maj to denote the assignment that one gets from the majority vote.

Theorem 5 *For d sufficiently large, there are some constants $c_1, c_2, c_3 > 0$ (where d and the c_i are independent of n), such that with probability at least $1 - n^{-c_1}$ over the choice of random formula F from the planted assignment model, F has a set S that is a t-core for $t > c_2 n$ with respect to maj , with the following two properties:*

1. *All variables of S are assigned by maj the same value as they are assigned by the planted assignment ϕ.*
2. *For every variable $x_i \notin S$, its propagation set in the graph $G_{S,\phi}$ satisfies $|P(x_i)| < c_3 \log n$.*

The condition $t > c_2 n$ implies that when n is sufficiently large, $t > c_3 \log n$. Theorem 5 together with Theorem 4 then imply that for large enough clause density d, for most formulas F with a planted assignment, our k-opt algorithm will find a satisfying assignment for F without need to increase k beyond $c_3 \log n$. Hence the running time of our algorithm will be polynomial in n.

A proof for Theorem 5 can be extracted from the analysis presented by Flaxman for his algorithm [11]. Specifically, the set of variables that remain assigned after Flaxman's unassignment phase may serve as the t-core S. Their assignment under maj agrees with the planted assignment ϕ. As to the unassigned variables, Flaxman considers a graph in which two unassigned variables are connected if they appear in the same clause, and shows that all its connected components are of size $O(\log n)$. Every edge of our graph $G_{S,\phi}$ is also an edge of Flaxman's graph (for the same S), implying that the connected components of $G_{S,\phi}$ are of size at most $O(\log n)$ as well. But this implies that also $|P(x_i)| = O(\log n)$. In Section A we sketch a self contained proof of Theorem 5.

5.1 The semirandom model

Recall that in our semi-random model, an adversary is allowed to add arbitrary 3CNF clauses of its choice, provided that all three literals in a clause are satisfied by the planted assignment ϕ. Flaxman's algorithm appears not to run in polynomial time in this semirandom model. Our algorithm does run in polynomial time.

Theorem 6 *For d sufficiently large, there are some constants $c_1, c_2 > 0$ (where d and the c_i are both independent of n), such that with probability* *at least $1 - n^{-c_1}$ over the choice of random formula F from the planted assignment model, regardless of the clauses added by the adversary in the semirandom model, our k-opt algorithm finds a satisfying assignment while maintaining $k \leq c_2 \log n$.*

Proof: If the adversary does not add any clauses, we have a t-core S with respect to maj as specified in Theorem 5. For all variables of S their assignment under maj agrees with their assignment under ϕ. Moreover, the graph $G_{S,\phi}$ does not have propagation sets larger than $c_2 \log n$. Consider now the addition of adversarial clauses. As all variables in these clauses are assigned consistently with ϕ, the majority vote for the semirandom formula still agrees with ϕ on S. Hence S remains a t-core. As to the graph $G_{S,\phi}$ associated with the semirandom formula, the adversary cannot add to it any edges compared to the original $G_{S,\phi}$, because with every directed edge (x_i, x_j) one needs to associate a clause in which x_i appears with a different polarity than its polarity in ϕ, but the adversary is not allowed to add any clause containing such a variable. Hence the size of the propagation sets cannot grow. It follows from Theorem 4 that the k-opt algorithm finds a satisfying assignment in polynomial time. □

6 Discussion

We presented a k-opt algorithm for 3SAT using the set-based landscape. The reason for using this landscape, rather than some other landscape such as the max-3SAT landscape, is that using it we can prove polynomial time convergence to a satisfying assignment in the planted assignment model, and in a semirandom variant of it. We do not know whether k-opt with the max-3SAT landscape has polynomial time convergence in the same model.

It is easy to design satisfiable 3CNF formulas on which our algorithm fails (or rather, takes exponential time to find a satisfying assignment). In particular, the k-opt paradigm is not expected to work well on formulas that have many local optima at large Hamming distance from the global optimum. We have performed preliminary experiments with our algorithm (in the implementation we followed some optimizations that are discussed in Section B in the appendix) on random formulas, where the number of variables ranged from $n = 20$ up to $n = 20,000$. In our experimentation we considered clause densities of $d = 3$ and $d = 4$ (instead of the larger clause densities that are assumed in Section 5), and formulas with or without planted assignments. Our goal was to check whether the average value of k reached by the algorithm appears to be a linear function of $\log_2 n$. For the planted assignment model, a plausible interpretation of our results is that $k \simeq 0.45 \log_2 n$ when $d = 3$, and $k \simeq 1.3 \log n$ when $d = 4$. When there was no planted assignment (note that even a random formula without a planted assignment is likely to be satisfiable at these clause densities), it appears as if $k \simeq 0.62 \log_2 n$ when $d = 3$. For $d = 4$, when $n = 100$ the average value of k was around 18, and for $n = 200$ the value of k typically exceeded 25, at which point we stopped the program (as its running time is exponential in k).

We remark that the algorithm presented here is one of two algorithms presented in [17] for the random and semirandom planted assignment models. The other algorithm started with a majority vote, continued with an unassignment phase, followed with unit clause propagation, and ended with a random walk-sat procedure. For details of this other algorithm and actual experimental results (on formulas with hundreds of thousands of variables and density 17), see [17].

Acknowledgements

This work was supported in part by a grant from the German-Israeli Foundation for Scientific Research and Development.

References

[1] D. Achlioptas, G.B. Sorkin. Optimal myopic algorithms for random $3SAT$. *IEEE Symposium on Foundations of Computer Science.* pp. 590-600, 2000.

[2] Mikhail Alekhnovich, Eli Ben-Sasson. Linear Upper Bounds for Random Walk on Small Density Random 3-CNFs. *FOCS 2003*, pp. 352-361.

[3] Noga Alon, Nabil Kahale. A spectral technique for coloring random 3-colorable graphs. *SIAM J. COMPUT.*, Vol. 26, No. 6,1733-1748. December 1997.

[4] Eli Ben-Sasson and Avi Wigderson. "Short proofs are narrow:resolution made simple". *Journal of the ACM (JACM)*, 48(2), 2001, 149–169.

[5] A. Blum, J. Spencer. Coloring random and semirandom k-colorable graphs. *J. Algorithms, 19*,pp. 204-234, 1995.

[6] R. Bryant. "Graph based algorithms for Boolean Function Manipulation". *IEEE Trans. Computers* 35(8), 677-691, 1986.

[7] Hui Chen, Alan Frieze. Coloring Bipartite Hypergraphs . *IPCO: 5th Integer Programming and Combinatorial Optimization Conference*, pp. 345-358, June 20, 2000.

[8] D.Du, J.Gu and P. Pardalos (Editors). *Satisfiability Problem: Theory and Applications.* DIMACS series in Discrete Mathematics and Theoretical Computer Science, AMS, 1997.

[9] Uriel Feige, Joe Kilian. Heuristics for Semirandom Graph Problems. *Journal of Computer and System Sciences 63*, pp.639-671, 2001.

[10] Uriel Feige, R. Krauthgamer. Finding and certifying a large hidden clique in a semi-random graph. *Random Structures and Algorithms 16(2)*, pp.195-208, 2000.

[11] Abraham Flaxman. A spectral technique for random satisfiable $3CNF$ formulas. *SODA 2003*, pp.357-363, July 2002.

[12] J. Friedman, A. Goerdt and M. Krivelevich. "Recognizing More Unsatisfiable Random 3-SAT Instances Efficiently". *Manuscript, 2003.*

[13] A. Frieze and C. McDiarmid. Algorithmic theory of random graphs. *Random Structures and Algorithms 10*, pp. 5-42, 1997.

[14] A. Kaporis, L. Kirousis, E. Lalas. The probabilistic analysis of a greedy satisfiability algorithm. *Proc. 5th Satisfiability Testing Workshop*, 362-376, 2002.

[15] E. Keydar. Finding Hamiltonian Cycles in Semi-Random Graphs. *MSc thesis.* The Weizmann Institue of Science, Rehovot, Israel, November 2002.

[16] C. Papadimitriou and K. Steiglitz. Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, 1982.

[17] Dan Vilenchik. "Finding a satisfying assignment for semi-random satisfiable 3CNF formulas". *MSc thesis.* The Weizmann Institue of Science, Rehovot, Israel, February 2004.

A Proofs

Here we sketch the proof of Theorem 5. Some of the missing details can be found in [17]. As noted in Section 5, a proof can also be deduced from the analysis in [11].

It is convenient for the sake of the proof to consider a slight variation on our random planted assignment model. Rather than select exactly dn clauses at random, one starts with all $7 \cdot \binom{n}{3}$ clauses that are satisfied by the planted assignment ϕ, and picks into F each such clause independently with probability $p = 3d/7\binom{n-1}{2}$. The expected number of clauses in F is thus dn, and with very high probability the actual number of clauses differs from the expectation only by low order terms. In our proofs, we shall switch freely between the original model (that we call F_d) and the new model (that we call F_p), according to which model is more convenient for the particular claim that we prove. Though there are minor technical differences between the models, they can be shown to be of negligible significance for our proofs.

As in previous sections, we use ϕ to denote the planted assignment, and *maj* to denote the assignment that one gets from the majority vote.

A.1 The majority vote

We first show that *maj* agrees with ϕ on almost all variables.

Proposition 7 *For any individual variable x_i, the probability that maj and ϕ disagree on it is $2^{-\Omega(d)}$.*

Proof: We prove this proposition in the F_p model. Assume w.l.o.g. that x_i appears positively in ϕ. Let t_i be a random variable counting the number of positive occurances of the variable x_i in the input formula F, and n_i a random variable counting the number of negative occurances. The majority vote is wrong with respect to x_i only if $n_i \geq t_i$. The total number of clauses that contain x_i positively and are satisfied by ϕ is exactly $4\binom{n-1}{2}$. The total number of clauses that contain x_i negatively and are satisfied by ϕ is exactly $3\binom{n-1}{2}$. The expectations of t_i and n_i are $E[t_i] = 4p\binom{n-1}{2} = 12d/7$, and $E[n_i] = 9d/7$. Each of t_i and n_i is a binomial random variable and hence strongly concentrated around its mean. In particular, $Pr[t_i \leq 11d/7] \leq 2^{-\Omega(d)}$ and $Pr[n_i \geq 10d/7] \leq 2^{-\Omega(d)}$. Hence $Pr[n_i \geq t_i] \leq 2^{-\Omega(d)}$. □

Proposition 8 *With high probability over the choice of input formula, the initial assignment maj disagrees with the planted assignment ϕ on at most $2^{-\Omega(d)}n$ variables.*

Proof: Proposition 7 together with the linearity of expectation implies that the expected number of variables on which *maj* and ϕ disagree is $2^{-\Omega(d)}n$. It follows from Markov's inequality (and fiddling with the constants in the Ω notation) that with probability $1 - 2^{-\Omega(d)}$ over the choice of random formula from the planted assignment model, *maj* and ϕ disagree on at most $2^{-\Omega(d)}n$ variables. This probability can be boosted up to $1 - 2^{-\Theta(n)}$ (where the constant in the Θ notation depends only on d) by considering the martingale implied by the clause selection process in the F_d model. Details are omitted (but can be found in [17]). □

Let us note that *maj* itself is not likely to be a satisfying assignment for F. A clause is not satisfied by *maj* if *maj* disagrees with ϕ on the assignment of the variables with which ϕ satisfies the clause (and agrees with ϕ on the assignment of the variables with which ϕ does not satisfy the clause). This is expected to happen for an $2^{-O(d)}$ fraction of the clauses.

A.2 The clause-sharing graph

To prove Theorem 5 we need to find a core for *maj* and analyse the propagation graph on the noncore variables. It is useful to consider first a certain *clause-sharing graph* G_F (or rather a multigraph, as we shall make no effort to avoid parallel edges). The vertices of the graph are the variables, and two variables are connected by an edge if there is a clause in F that contains both of them. As each clause contributes three edges to G_F and the number of clauses is dn, the average degree in G_F is $6d$. In our analysis we shall encounter various vertex induced subgraphs of G_F. The constants in the following lemma are to some extent arbitrary.

Lemma 9 *For d sufficiently large (e.g., $d = 30$) there is some constant $c > 0$ (e.g., $c = 1/500$) such that with high probability over the choice of F, every vertex induced subgraph of G_F with average degree at least $d/7$ contains at least cn vertices.*

Proof: We prove the proposition in the F_p model. Let S be a set of variables such that $G_F(S)$, the subgraph induced on S, has average degree at least $d/7$. Then $G_F(S)$ contains at least $d|S|/14$ edges, implying that F has at least $f = d|S|/14$ clauses that contain two variables from S. (If a clause contains three variables from S it contributes three edges to $G_F(S)$, and hence f may be in fact smaller. A rigorous proof should take this fact into acount, but it turns out that this has only negligible effect on the final outcome, and for simplicity, we ignore this issue here.) These f clauses are chosen from at most $g = 7n\binom{|S|}{2}$ clauses, so the probability that $G_F(S)$ has average degree at least $d/7$ is at most $\binom{g}{f}p^f$. Taking the union bound over all sets S of all sizes up to cn, one gets that the probability of the proposition failing is at most roughly

$$\sum_{i=d/7}^{cn} \binom{n}{i} \binom{7ni^2/2}{di/14} (6d/7n^2)^{di/14} \simeq \sum_{i=d/7}^{cn} \binom{n}{i} (42e \cdot i/n)^{di/14}$$

which tends to 0 at a rate polynomial in n, when d is sufficiently large and c sufficiently small. □

A.3 The core

We shall identify here a particular t-core that is likely to exist for F.

Definition 2 *For formula F, assignment ψ and set S of variables, we say that variable x_i* supports *clause C_j with respect to (ψ, S) (or with respect to ψ, if S contains all variables) if all three variables in C_j are from S, C_j contains x_i, the occurence of x_i in C_j is assigned to true under ψ, and the two other literals in C_j are assigned to false under ψ. For a nonnegative integer q, a variable is a q-*supporter *with respect to (ψ, S) if there are at least q clause that it supports with respect to (ψ, S).*

The notion of supporters will help us define a core. We shall first consider supporters with respect to the planted assignment ϕ, and then with respect to maj .

Proposition 10 *With high probability over the choice of F, all but a fraction of $2^{-\Omega(d)}$ of the variables are $2d/7$-supporters with respect to the planted assignment ϕ.*

Proof: We first consider the F_p model, and all possible clauses. x_i is a supporter with respect to ϕ of $\binom{n-1}{2}$ clauses. Each such clause is picked into F with probability $p = 3d/7\binom{n-1}{2}$. Hence x_i is expected to support $3d/7$ clauses, and has probability $2^{-\Omega(d)}$ of supporting less than $2d/7$ clauses. The expected number of variables that are not $2d/7$-supportes with respect to ϕ is $2^{-\Omega(d)}n$. The actual number is similar with very high probability, as can be shown via a martingale argument over the clause exposure process in the F_d model. Details are omitted. □

Definition 3 *A set S of variables is a* supporting set *with respect to maj if every variable in S is a $d/7$-supporter with respect to $(maj\ , S)$.*

Proposition 11 *With high probability over the choice of F, its supporting set with respect to maj is nonempty.*

Proof: We consider the following procedure for identifying a supporting set S. Initially, S contains all variables that are $2d/7$-supporters with respect to ϕ. Then we remove from S all variables on which ϕ and maj disagree. Then iteratively, remove from S those variables that are not $d/7$-supporters with respect to $(maj\ , S)$, updating S after every removal. We shall show that with high probability over the choice of F, the above process ends with a nonempty S.

Let A be the set of variables that are not $2d/7$-supporters with respect to ϕ, B the set of variables on which ϕ and maj disagree, and C the set of variables removed in the iterations. Then $|S| \geq n - |A| - |B| - |C|$. Proposition 10 implies that $|A| < 2^{-\Omega(d)}n$. Proposition 8 implies that $|B| < 2^{-\Omega(d)}n$. We show that with high probability, $|C| \leq |A| + |B|$. Assume to the contrary that at some stage C reached a cardinality of $|A|+|B|$, and consider the subgraph of the clause-sharing graph induced on $A \bigcup B \bigcup C$. It has at most $2|C|$ vertices and least $|C|d/7$ edges (and hence average degree $d/7$) because every variable in C supports (with respect to ϕ) at least $2d/7 - d/7$ clauses that involve variables from $A \bigcup B \bigcup C$. But then, by Lemma 9, $2|C| > cn$, which for sufficiently large d contradicts our choice of $2|C| \leq 2(|A|+|B|) \leq 2^{-\Omega(d)}n$. □

The supporting set S found in the proof of Proposition 11 has the following two useful properties:

1. It satisfies $|S| \geq (1 - 2^{-\Omega(d)})n$.
2. The assignments ϕ and maj agree on every variable in S.

This leads to the following.

Proposition 12 *The supporting set S can serve as a t-core for maj , with $t = \Omega(n)$.*

Proof: We have seen already that ϕ and maj agree on S, hence the subformula F_S induced on S is satisfied by maj . It remains to show that every other assignment that satisfies F_S differs from maj on $\Omega(n)$ variables of S. Intuitively, this is true for the following reason. Flipping one variable in S causes the $d/7$ clauses that it supports in F_S to be not satisfied. This requires flipping of additional variables in S, and again, the clauses that these variables support in F_S become not satisfied. This causes an "avalanche effect" that stops only after $\Omega(n)$ variables are flipped.

Formally, suppose a set $T \subset S$ of variables were flipped and F_S remained satisfied. Each of the flipped variables supports at least $d/7$ clauses, and in each such clause some additional variable must belong to T. Hence the subgraph of the clause-sharing graph induced on T has minimum degree at least $d/7$, and Lemma 9 implies that $|T| > cn$. □

A.4 The propagation sets

Let S be a core for maj as found by Proposition 12. We shall prove that the propagation sets in the propogation graph $G_{S,maj}$ are all of size at most $\log n$.

Our intention is to prove the claim by using the following approach. Consider an arbitrary set K of $\log n$ variables (say $x_1, \ldots, x_{\log n}$) one of which (say x_1) is called the *root* variable. On this set of variables let us consider a directed tree structure T_K with x being at the root of the tree, and the other variables of K serving each as either an internal vertex or a leaf of the tree. We say that a 3CNF formula F with planted assignment ϕ induces T_K if for every directed tree edge (x_i, x_j), F contains a clause in which x_i appears in polarity disagreeing with ϕ and x_j appears in polarity agreeing with ϕ. With T_K we associate two random events. $T_K \subset F$ is the event that F induces T_K (taking into account also ϕ). The other event is $K \subset \bar{S}$, where S is the core set.

Proposition 13 *The propagation graph $G_{S,maj}$ contains a propagation set of size $\log n$ only if for some T_K as above, both events $T_K \subset F$ and $K \subset \bar{S}$ hold.*

Proof: Assume that the propagation set of variable x_i satisfies $|P(x_i)| \geq \log n$. Remove variables from $P(x_i)$ (starting from those furthest away from x_i) until a set $P'(x_i)$ of exactly $\log n - 1$ variables remains. Now let $K = x_i \bigcup P'(x_i)$, and let T_K be the directed tree in $G_{S,maj}$ that shows that $P'(x_i)$ is in the propagation set for x_i. Then by the definition of the propagation graph, T_k is necessarily induced by F, and $K \subset \bar{S}$. □

Proposition 14 *Fixing K and T_K, $Pr[T_K \subset F] \leq (O(d/n))^{\log n}$, where the probability is computed over the choice of ϕ and F.*

Proof: Consider a minimal set of clauses I that can induce T_K (by containing variables in K with the right pairing and polarity as required by T_K). Note that $|I| \leq |K| - 1$ because of minimality, and $|I| \geq (|K| - 1)/2$ because each clause gives at most two edges of T_K. The following argument can be performed separately for each possible value of $l = |I|$. For every directed edge in T_K, there are at most $2n$ different clauses that may possibly induce it (depending on the identity of the third literal). Hence there are at most $(2n)^l \binom{|K|-1}{l} = (O(n))^l$ ways of choosing I. The probability that a particular set I is in F (in the F_p model) is $p^{|I|} = (O(d/n^2))^l$. The union bound over all I now proves the proposition. □

Proposition 15 *Fixing K, $Pr[K \subset \bar{S}] \leq 2^{-\Omega(d \log n)}$, where the probability is computed over the choice of ϕ and F.*

Proof: The size of $\bar{S}$ is $2^{-\Omega(d)}n$. (For simplicity, we assume here that this holds with certainty. More details will follow later.) Moreover, the set of variables in $\bar{S}$ is random (since all distributions in questions are invariant under renaming of variables). As $|K| = \log n$, $Pr[K \subset \bar{S}] \leq 2^{-\Omega(d \log n)}$. □

For a specific T_K, what is the probability that both events $T_K \subset F$ and $K \subset \bar{S}$ hold? If these events were independent, the probabilty would be at most $(O(d/n))^{\log n} \cdot 2^{-\Omega(d \log n)}$. Then we could consider all $\binom{n}{\log n} \simeq (\frac{en}{\log n})^{\log n}$ possible sets K, multiplied by the number of possible trees on K (which is less then $(\log n)^{\log n}$), and use the union bound to conclude that with high probability, there is no T_K for which both events $T_K \subset F$ and $K \subset \bar{S}$ hold. Then by Proposition 13 we would reach the desired conclusion that there are no propagation sets larger than $\log n$.

The main problem in the approach outlined above is that the events $K \subset \bar{S}$ and $T_K \subset F$ are not independent, and hence the probability that they both happen is not the product of their individual probabilities. We shall now explain how these dependencies can be handled, and in doing so, also handle some other minor inaccuracies in the arguments given above. The approach we follow is patterned after the approach used in [3, 11, 17], though the details are a bit different.

For a planted assignment ϕ and a propagation tree T_k, let I be a minimal set of clauses that may potentially induce T_K. I contains at most $K - 1$ clauses, and hence at least $|K|/2$ variables of K appear at most 6 times in I. Denote the set of these variables by K'.

In what follows, we condition the analysis on the event that the terms of Lemma 9 indeed hold for the 3CNF formula F. The conditioning is justified because this event happens with high probability.

We now use the following approach to decouple the dependency between $T_K \subset F$ and $K \subset \bar{S}$. Fix the planted assignment ϕ, and consider a set K of variables, a propagation tree T_K consistent with ϕ, and a minimal set I of clauses that can induce T_K. Now think of F as being generated in the model F_p in two stages. In the first stage, we generate F' by including every eligible clause (satisfied by ϕ) with probability p, except that the clauses of I are not included in the process of generating F'. For F' we define an *inner core* S'. The inner core is obtained by providing some "padding" to the process described in Proposition 11 for generating a core. For the majority vote, this padding means that a variable is set in *maj* to agree with the planted assignment ϕ only if the majority vote had a bias of more than 6 towards the assignment in ϕ. For the definition of supporting sets, the padding means that each variable needs to support more than $d/7$ clauses, *but we do not count clauses that contain two variables that appear in I*. It is not hard to see that all proofs of sections A.1 and A.3 go through even when this padding is applied (though perhaps with some minor differences in the choice of constants). Moreover, the variables that appear in I are indistinguishable in these proofs from the variables not I, because I contains only $O(\log n)$ clauses, and in expectation would not contribute even a single clause to F. Hence there is an inner core S' of size $(1 - 2^{-\Omega(d)})n$, and each variable of K' (note that here we will be considering K' rather than K) is not is S' with probability $2^{-\Omega(d)}$. Moreover, conditioned on Lemma 9, the probability of such an S' not existing is $2^{-\Omega(n)}$, where the hidden constant in $\Omega(n)$ depends only on d. (The martingale arguments in Propositions 8 and 10 have such low error probability. Lemma 9 has higher error probability, and this is why we condition on it.)

Now we reach the second stage for generating F, in which each clause of I is placed in F with probability p. The main point is that now, regardless of the outcome of the second stage, every variable of K' originally in the inner core S' is necessarily a member of the core S for F, as the set of clauses I can neither tilt the majority vote for such a variable, nor effect the number of clauses it supports. (This claims follow from the padding that we had for S'). Hence we may in fact take the products of the probabilities of the events $K' \subset \bar{S}'$ and $I \subset F$ as an upper bound on the probability that both events $K \subset \bar{S}$ and $I \subset F$ happen. (An argument similar to Proposition 14 ties between the events $I \subset F$ and $T_K \subset F$.) Following the principles and calculations as outlined above, we have:

Proposition 16 *With high probability, the propagation graph induced on the noncore variables does not have a propagation set larger than* $\log n$*.*

The combination of Propositions 12 and 16 proves Theorem 5.

B More discussions

Some notes about actual implementation of the algorithm. We chose an initial assignment based on the majority vote. This choice was motivated by the particular random model that we used for generating input formulas. If the algorithm is run on instances drawn from a different distribution, one should consider other methods of producing an initial assignment. For example, one may start from a random initial assignment, or one that is derived using spectral techniques as in [3, 11]. Another issue is how to arbitrate among improving moves if several such moves are available. Here again, some prior knowledge about the structure of the formula may help. A simple principle that can be used is to sort the variables in decreasing order of confidence in their current assignment, and use this order when labeling the edges leading out of the root of the search tree. This confidence level can be computed anew after every step, or more simply, only once following the choice of the initial assignment. For example, in the case of an initial assignment based on majority vote, the variables may be sorted in increasing order of the statistical confidence in their majority vote.

A natural line of research to follow is to consider other semi-random models, and see if the algorithm still works (or else, design an improved algorithm). Let us mention here a few semirandom models that we find potentially interesting. In all of them, the adversary does not add clauses, but only flips the polarity of existing variables.

1. **Positive flips.** The adversary may flip the polarity of the occurence of a variable that does not agree with the planted assignment ϕ so that it does agree with ϕ. This can only help the initial majority vote, but can greatly damage the core. A variable x_i originally in the core can be thrown out of the core by the adversary flipping the polarity of other variables in the clauses that x_i supports. Hence our current analysis for k-opt does not work, though potentially something else does work.

2. **Negative flips.** The adversary may flip the polarity of the occurence of a variable that does agree with the planted assignment ϕ so that it does not agree with ϕ, provided that the clause remains satisfied. This might ruin the strong correlation between an initial majority vote and the planted assignment ϕ. On the other hand, all original variables of the core remain in the core (w.r.t. to the original majority vote assignment, prior to the adversary's intervention). Possibly, an initial assignment based on spectral techniques may replace here the majority vote assignment.

3. **Arbitrary flips.** Only the variables in each clause are chosen at random. The polarity is chosen by an adversary, under the constraint that all clauses are satisfied by the planted assignment ϕ. It is our belief that currently known algorithmic techniques are not strong enough to handle this semi-random model (except when the density is very small).