Search Rearrangement Backtracking
Often Requires Exponential Time
To Verify Unsatisfiability

By

John Franco
Department of Computer Science
Indiana University
Bloomington, IN 47405

TECHNICAL REPORT NO. 210

Search Rearrangement Backtracking
Often Requires Exponential Time
To Verify Unsatisfiability

by

John Franco

December 10, 1986 (Revised July 5, 1987)

This report is based on work supported in part by the Air Force Office of Scientific Research under Grant No. AFOSR-84-0372.

ABSTRACT

It is shown that any form of Search Rearrangement Backtracking (SRB) requires exponential time to verify the unsatisfiability of nearly all of a wide class of CNF boolean expressions. This result is based on an input model which generates n independent k-literal clauses from a set of r boolean variables. We assume that k is fixed and n and r tend to infinity. The result holds if $\lim_{n\to\infty} n/r(n) = \lambda$, is fixed and $\lambda > \ln(2)/(-\ln(1-2^{-k}))$. We also show that SRB requires superpolynomial time nearly always if λ is replaced by $\lambda(n) = o(n^{1/\ln\ln(n)})$ and $\lim_{n\to\infty} \lambda(n) = \infty$ (so the superpolynomial time result holds, for example, if $\lambda(n) = (\ln(n))^{\beta}$ where β is any positive constant). We also show that these results apply to any form of the Davis-Putnam Procedure.

1. Introduction

The Satisfiability problem (SAT) is the problem of determining whether there exists an assignment of values to boolean variables (a truth assignment) which causes a given boolean expression I to have value *true*. A truth assignment which causes I to have value *true* is called a satisfying truth assignment and is said to satisfy I. It is well known that SAT is NP-hard. However, it has been shown that some algorithms solve SAT efficiently in a probabilistic sense under certain conditions. These conditions are determined by the parameters of the input models chosen for analysis.

The model we consider in this paper and denoted $M(n,r,k)$ is the constant clause size model for CNF boolean formulas. An Instance of SAT generated according to $M(n,r,k)$ is the conjunction of n k-literal clauses (disjunctions) each selected uniformly and with replacement from the set of all k-literal clauses that can be composed from r boolean variables with the property that no two literals in a clause are associated with the same variable. We will assume that k is fixed (independent of n and r), and we use $k-$SAT in place of SAT when referring to instances generated by M. The problem of finding a satisfying truth assignment for an instance of k-SAT or verifying that no such truth assignment exists is NP-hard if $k \geq 3$ ([12]). The model $M(n,r,k)$ has the interesting property [10] that if $n/r < \ln(2)/(-\ln(1-2^{-k}))$ then the average number of truth assignments that satisfy random instances of k-SAT is exponential in r and if $n/r > \ln(2)/(-\ln(1-2^{-k}))$ then almost all random instances have no satisfying truth assignments (that is, they are unsatisfiable).

Surprizingly simple and fast algorithms are very effective at finding satisfying truth assignments when at least one exists if instances of k-SAT are generated according to $M(n,r,k)$. For example, the unit-clause algorithm is: repeatedly assign values arbitrarily to variables in random order until some clause has just one non-falsified literal (a unit clause), then assign to the variable associated with that literal the value which satisfies the unit clause and repeat these two steps until all variables have been assigned values. In [5] it is shown that the unit-clause algorithm finds a satisfying truth assignment for a random instance of k-SAT with bounded probability under $M(n,r,k)$ if $n/r < (2^{k-1}/k)((k-1)/(k-2))^{k-2}$. A generalization of the unit-clause heuristic (choose some variable that appears in a smallest clause and assign to it the value which satisfies that clause) is shown in [5] to find a satisfying truth assignment to almost all random instances of k-SAT if $n/r < (.46 * 2^k/(k+1))((k-1)/(k-2))^{k-2} - 1$ and $4 \leq k \leq 40$ and if $40 < k$ and $n/r < 10^{10}$ (for practical purposes this is all ratios of n/r). The following algorithm is an improvement to the unit-clause algorithm: repeatedly assign a value which satisfies a unit clause or, if no unit-clauses exist, a value which satisfies most remaining clauses (instead of assigning values arbitrarily). In [4] it is shown that this improvement results in an algorithm that finds a satisfying truth assignment for a random instance of 3-SAT with bounded probability under $M(n,r,3)$ when $n/r < 2.9$. Similar results have been obtained for other instance distributions (see, for example, [9], [13] and [15]).

Although the algorithms of the previous paragraph involve no backtracking, the heuristics for choosing variables and values presented in those algorithms can easily be incorporated into a backtrack structure. In fact, the Davis-Putnam Procedure (DPP), given in [6] and [7], includes the unit-clause heuristic. Thus, the results mentioned in the previous paragraph apply directly to DPP in the case that it stops when a satisfying truth assignment is obtained. The heuristics employed by DPP are important since without them DPP would almost always require time exponential in r for any fixed ratio of n to r ([10], [11]). In this paper we investigate how important these and other heuristics are to backtracking when we are interested in verifying that no satisfying truth assignment exists for an unsatisfiable instance of k-SAT. The class of heuristics we consider, when applied to simple backtracking, produces the class of problem solving procedures known as Search Rearrangement Backtracking (SRB) and discussed in [2], [14], [16] and [17].

We present SRB as an algorithm in which clauses are represented as sets of literals and instances are represented as collections of clauses. In this representation, under a partial assignment of values to variables, a *false* literal is removed from a clause and a *true* clause is removed from an instance. Let $H(I)$ be a function that maps instances I of SAT to boolean variables in I. The H function represents a wide class of heuristics for dynamically choosing the next variable to eliminate in a backtrack search. We use the convention that the positive literal associated with variable v is denoted v and the negative literal associated with variable v is denoted $\bar{v}$. Search Rearrangement Backtracking applied to I is

```
SRB(I):
    If I has a null clause then return "UNSAT"
    Else if I is empty then return "SAT"
    Else
        $v \leftarrow H(I)$
        $I_1 \leftarrow \{c - \{\bar{v}\} : c \in I, v \notin c\}$
        $I_2 \leftarrow \{c - \{v\} : c \in I, \bar{v} \notin c\}$
        If SRB($I_1$)="UNSAT" and SRB($I_2$)="UNSAT" then return "UNSAT"
        Else return "SAT"
```

In SRB, I_1 is the subinstance of SAT obtained from I by assigning the value *true* to variable v and I_2 is the subinstance obtained by assigning the value *false* to v. Although it is not necessary to do so, we have restricted H by forbidding it to choose the same variable twice in the same branch of a backtrack search tree. A computation in which a variable is selected twice in the same branch can always be transformed to a shorter computation in which no variable is selected twice in the same branch. Therefore our lower bounds using restricted H functions apply to all H functions. Our H function is such that SRB does not include algorithms where the choice of v is determined randomly. However, there is a best $H(I)$ for every I (which minimizes the time to develop a refutation for I) and this H will perform no worse than any randomized method for choosing v. Therefore, our result provides a lower bound for randomized methods.

Note that SRB is actually the class of all backtracking algorithms for instances of SAT which invoke backup when some clause has become falsified or a satisfying truth assignment has been found. Each algorithm in the class is distinguished by its H function which may cause dynamic or static variable elimination, and need only return a value in finite time. An interesting class of backtracking algorithms, known as Multi-Level Search Rearrangement Backtracking algorithms, were the inspiration for this paper. These algorithms, as analysed in [2], [14], [16] and [17], for example, fall into the class SRB if the computational effort required to evaluate potential variable eliminations is allowed to show up as part of the search tree. The H functions associated with these algorithms are fairly complicated and involve looking many levels deeper into the search tree to pick the next variable elimination which is most likely to result in a small subtree. The results of [2] and [17] are that the H function has a significant impact on search tree size. For example, in [17], 3-SAT instances containing 4096 clauses composed from 256 variables ($n/r = 16$) were solved by Two-Level Search Rearrangement Backtracking using 10^{-17} as many search tree nodes, on the average, as ordinary backtracking.

The most important previous work on average case analysis of backtracking algorithms for the Satisfiability problem using the model $M(n,r,k)$ appears in [3] and [16]. The algorithms of both papers search for all solutions to a given input. In [3] it is shown that, under $M(n,r,k)$, if $n/r = r^{\alpha-1}$ where $1 < \alpha < k$, α constant, then ordinary backtrack trees contain at least $e^{\Theta(r^{(k-\alpha)/(k-1)})}$ nodes on the average. Recall that, under the condition of the previous sentence, almost all inputs have no solutions. Therefore, the result of [3] implies that ordinary backtracking requires exponential average time to verify unsatisfiability if $n/r = r^{\alpha-1}$, $1 < \alpha < k$, but the exponential is sublinear and decreases with α until $\alpha = k$. When $\alpha \geq k$ (that is, $n/r \geq r^{k-1}$) then results in [3] imply that ordinary backtracking requires polynomial average time. In [16] the same kind of results are obtained for Simple Search Rearrangement Backtracking. In particular, over the range $1 < \alpha < k-1$, Simple-Search-Rearrangment-Backtrack trees contain at least $e^{\Theta(r^{(k-\alpha-1)/(k-2)})}$ nodes on the average. Thus, Simple Search Rearrangement Backtracking has exponentially better average case performance than ordinary backtracking if $n/r = r^{\alpha-1}$, $1 < \alpha < k-1$.

The algorithms of [3] and [16] may be regarded as specific forms of SRB (that is, specific H functions) if the "look-ahead" effort is taken into account. This paper shows that no matter how clever the H function, even if it is vastly improved over another H function, it will not be clever enough to yield polynomial average time on almost all unsatisfiable instances of k-SAT if $n/r(n) = o(n^{1/\ln\ln(n)})$ and $n/r(n) > \ln(2)/(-\ln(1-2^{-k}))$ for all $n > 0$.

We are interested in the performance of SRB when inputs are almost always unsatisfiable; that is, when inputs are generated according to $M(n,r,k)$ and $n/r > \ln(2)/(-\ln(1-2^{-k}))$. We prove that all algorithms in the class SRB require time exponential in r almost always when $n/r = \lambda$ where λ is fixed and is greater than $\ln(2)/(-\ln(1-2^{-k}))$. The proof itself is interesting because it relies on a *structural* property of instances of k-SAT which must be present in almost all random instances and cannot be present if the search tree corresponding to the execution of SRB on unsatisfiable instances is small. The property, loosely speaking, is that the number of pairs of clauses containing literals associated with the same variable is small if $n/r = \lambda$ for any fixed λ. Although unsatisfiable instances of k-SAT are generated under $M(n,r,k)$ when $n/r < \ln(2)/(-\ln(1-2^{-k}))$, we are unable to use the property mentioned to extend the results to that range because almost all instances in that range are satisfiable (so it is possible that almost all instances have the property but almost no unsatisfiable instances do). We also show that SRB requires superpolynomial time if $n/r = o(n^{1/\ln\ln(n)})$ and $\lim_{n\to\infty} n/r = \infty$.

We also show that the same result applies to any form of DPP. DPP looks for unit-clauses and pure-literals. For our purposes, a pure-literal is a variable which appears only as a positive literal or as a negative literal in I. DPP is like SRB in the sense that there is some heuristic function H which selects the next variable to be assigned a value. The heuristic function of DPP selects a pure-literal next or, if no pure-literals are present, a unit-clause next or, if no unit-clauses are present, a variable with highest "weight". DPP differs from SRB in that either I_1 or I_2 but not both is used as a recursive argument to DPP when the selected variable is a pure-literal in I. It is this feature of DPP that prevents us from directly applying the results to DPP. However, we will show how to design an SRB algorithm from a given DPP algorithm which runs faster (generates fewer nodes) than the DPP algorithm if the given instance is unsatisfiable. This implies that the result also holds for DPP.

The results we get are pessimistic and are possibly surprizing to those familiar with a result of Purdom [14] which states that even ordinary backtracking can verify unsatisfiability in polynomial time, on the average, for a variety of relationships between model parameters. Purdom's *random clause* model and model M have certain similarities. In both models n clauses are independently constructed from r boolean variables. But, in the random clause model, instead of a fixed number of literals per clause, each literal appears independently in a clause with probability p. Thus, if we set $2pr = k$, clauses have k literals, on the average, as for model M. Also, if $n/r > \ln(2)/(-\ln(1-(1-p)^r))$ then almost all instances are unsatisfiable. If $2pr = k$ this condition is nearly $n/r > \ln(2)/(-\ln(1-e^{-k/2}))$ which is similar to the condition that almost all instances are unsatisfiable in model M. The results of Purdom's work in this area are that various backtracking algorithms exhibit very different average case behavior depending on the values given to parameters. Purdom's results are interesting (and parallel our own results) because they show that these algorithms are fast on the average if instances are usually "very" unsatisfiable, are slow if instances are "moderately" unsatisfiable or "moderately" satisfiable (not too many literal links between clauses), and fast if instances are "highly" satisfiable. Model M also has this property. When $n/r(n) \geq (r(n))^{k-1}$, or $n/r(n) \leq \ln(n)/n$ there are SRB algorithms that almost always solve problems in polynomial time (see [3] and [15]). This paper is concerned with the range $\ln(2)/(-\ln(1-2^{-k})) \leq n/r \leq o(n^{1/\ln\ln(n)})$, which is in the intermediate region for model M.

A result of [14] is that if $2pr = k$, k fixed, then for n/r big enough, ordinary backtracking requires polynomial time, on the average. This result appears to be strikingly different from ours but can be accounted for in the following way. Under the random clause model, the probability that a random instance contains a null clause (no literals) is $1-(1-(1-p)^{2r})^n$ which is $1-(1-e^{-k})^n$ in the limit if $2pr = k$. But, if a null clause appears in the given instance, backtracking stops and states the given instance is unsatisfiable without doing any searching at all. The time required by backtracking in that case is the time to locate a null clause which is $O(n)$ at worst. If the time required by backtracking to verify the unsatisfiability of instances that do not originally contain a null clause is $(1-e^{-k})^{-n}$ (that is, exponential in n) then the average time required by backtracking is $O(n)\cdot(1-(1-e^{-k})^n)+(1-e^{-k})^n(1-e^{-k})^{-n} = O(n)$. Thus it is possible that all or nearly all random instances with no null clauses which are generated under the random clause model with $2pr = k$ are solved in exponential time by backtracking and yet the average time for backtracking is polynomial in n. Model M does not generate any null clauses so our result is not inconsistent with polynomial average time under the random clause model not only for ordinary backtracking but for more sophisticated forms of backtracking such as the algorithm in [17]. Another way to look at the relationship between our result and the results under the random clause model is to regard model M as generating a very small and non-easy subset of the instances that the random clause model generates.

2. Analysis

We use a binary tree, denoted $T_I(H)$, to model the execution of SRB for a particular H function on a given instance I of k-SAT in the customary manner. Associated with each non-leaf x in $T_I(H)$ is a boolean variable $v(x)$ contained in I and a subinstance $I(x)$ of I. Associated with the edge connecting x to its left (right) child is the interpretation that $v(x)$ is assigned the value *true* (*false*), respectively. Associated with a path from the root of $T_I(H)$ to any node x is the partial assignment $P(x)$ of values to the variables corresponding to nodes visited on that path except for x. Specifically, $P(x)$ is a set of variable/assignment pairs $(v \leftarrow t)$, one pair for each v associated with a node on the path from the root down to but not including x, where t is *true* (*false*) if the left (right) son of the node associated with v is on the path to x. We defer a discussion on the meaning and determination of $I(x)$ until we develop some intuition about $T_I(H)$.

Associated with leaves and edges of $T_I(H)$ are labels corresponding to clauses in I. Let each clause in I be given a name that uniquely identifies that clause. We label every leaf x of $T_I(H)$ with the name of the clause that is null under $P(x)$, if at least one clause is null under $P(x)$. If more than one clause is null under $P(x)$ then x is labeled with the name of one of the null clauses arbitrarily. If no clauses are null under $P(x)$ then x is given no label. We associate with each edge of $T_I(H)$ a set of literal/clause-label tuples (called edge labels) as follows. Let x be a non-leaf of $T_I(H)$ with left child y and right child z. Let $\langle x, y\rangle$ be the edge connecting x to y, let $\langle x, z\rangle$ be the edge connecting x to z. Let $v(x)$ be the variable associated with x. If l is a label given to a leaf of the subtree of $T_I(H)$ having y (alternatively z) as root, and the literal $\bar{v}(x)$ (alternatively $v(x)$) is in the clause corresponding to l then $(v(x), l)$ (alternatively $(\bar{v}(x), l)$) is a member of the list of labels associated with $\langle x, y\rangle$ (alternatively $\langle x, z\rangle$). If l is in an edge label associated with edge e then, for brevity, we say l is associated with e.

Figure 1 contains an example of the labeling of leaves, association of labels to edges, and the association of variables and subinstances to nodes of a subtree of $T_I(H)$ rooted at x given the instance

$$I = (\bar{v}_1, \bar{v}_2, \bar{v}_4)(\bar{v}_3, v_5, v_6)(\bar{v}_2, v_3, v_4)(v_2, v_5, v_6)(v_1, \bar{v}_2, v_3)$$

and some H function where x is such that $P(x) = \{(v_5 \leftarrow false), (v_6 \leftarrow false)\}$. To simplify the figure we have shown only the clause labels associated with edges: the literals of the edge labels that are actually associated with edges are implied. If x is a leaf of $T_I(H)$ and l labels x, then the label l is associated with exactly k edges on the path from the root of $T_I(H)$ to x (namely those edges which represent the partial truth assignment which falsifies all k literals in the clause labeled l). If x is not a leaf and has left child y and right child z and the edge label $(\bar{v}(x), l)$ is associated with the edge $\langle x, y\rangle$ then l

cannot be associated with any edge in the subtree of $T_I(H)$ rooted at z for the following reasons: (1) $v(x)$ cannot be associated with any node below and including z, (2) the clause labeled l must contain $v(x)$, and (3) complementary literals are not allowed in the same clause (so $(v(x), l)$ cannot be associated with $\langle x, z\rangle$). Consequently, l cannot be associated with any leaf under z. Similarly, if the edge label $(v(x), l)$ is associated with $\langle x, z\rangle$ then l cannot be associated with any edge in the subtree of $T_I(H)$ rooted at y and cannot label a leaf under y.

We now define $I(x)$, a subinstance of I associated with node x. $I(x)$ is the subset of clauses in I that label leaves below x. Note that even if a clause labels many leaves below x it appears only once in $I(x)$. Also note that clauses in $I(x)$ have exactly k literals.

In this paper we are concerned with verifying unsatisfiability. If an instance is unsatisfiable then $T_I(H)$ is a refutation tree and has the property that all its leaves are labeled since backtracking is only due to the emergence of a null clause. From now on it will be understood that $T_I(H)$ is a refutation tree. We make the following simplifying assumption:

Assumption S:

At least one label is associated with every edge of $T_I(H)$.

If there exists an edge $\langle x, y\rangle$ with no associated edge labels then a computation involving fewer nodes is possible: simply replace the subtree rooted at x with the subtree rooted at y. Thus, lower bounds derived from the simplifying assumption will apply to all search strategies. The effect of this assumption is to allow backtracking using the pure-literal rule. We will see this more clearly at the end of the next section when we consider the Davis-Putnam Procedure.

Each clause label associated with a leaf of $T_I(H)$ must also be in edge labels associated with k edges on the path from the root of $T_I(H)$ to that leaf or else some clause labeling a leaf is not falsified by the truth assignment associated with that leaf. Furthermore, if a clause labels more than one leaf then the k edges on each path from the root to such a leaf must all be associated with the same k variables and assignments (namely those that make the clause $false$). Thus, for any node x in a refutation tree such that $p(x)$ distinct leaf labels exist below x, the number of distinct edge labels in $T_I(H)$ due to those leaf labels is $kp(x)$. For every node x let $c(x)$ be the number of distinct edge labels at or below x and define $h(x) = kp(x) - c(x)$, the number of edge labels above x which are due to leaf labels below x. Clearly, if $h(x) > 0$ then x cannot be the root of a refutation tree. In Figure 1, $h(x) = 3 * 5 - 11 = 4$ with specific contributions from (v_5, c_2), (v_6, c_2), (v_5, c_4) and (v_6, c_4) which are the edge labels above x that are due to the clauses labeling leaves below x. We will show that $h(x) > 0$ in a small tree, with probability tending to 1. This will be used to show that $T_I(H)$ cannot be small, with probability tending to 1.

We also introduce a function $common(x)$. Let x be a node in $T_I(H)$, let $c(x)$ be the number of distinct edge labels at or below x, and let $var(x)$ be the number of distinct variables associated with nodes at or below x. We define $common(x) = c(x) - var(x)$. For example, in Figure 1, $c(x) = 11$ and $common(x) = 7$.

Finally, we define a function $Icomm(I')$. Let I' be any collection of clauses. Then $Icomm(I')$ is the total number of literals in I' that are associated with variables that appear two or more times in I' minus the number of distinct variables of that kind. $Icomm(I')$ is also the total number of literals in I' minus the number of distinct variables in I' since there is exactly one literal in I' for every variable that appears once in I'. Note the similarity between $Icomm$ and $common(x)$. However, $Icomm$ is defined for any collection of clauses and not just those sets of clauses corresponding to $I(x)$ where x is a node of $T_I(H)$. We even allow $Icomm(I')$ to be defined if one or more clauses in I' contain duplicate or complementary literals. We make use of $Icomm$ to bound $common(x)$ in the following way. Let I be any instance of k-SAT. For any H function, let x be a node in $T_I(H)$ and suppose $I(x)$ contains $p(x)$ clauses and $common(x) > p(x)(1 + 1/\ln(p(x)))$. Then there is a subset I' of I containing $p(x)$ clauses such that $Icomm(I') > p(x)(1 + 1/\ln(p(x)))$, namely all the clauses of $I(x)$. Hence we may make the following

Observation 1:

Suppose that no subset I' of clauses in an instance I of k-SAT which contains p clauses is such that $Icomm(I') > p(1 + 1/\ln(p))$. Then, for all H functions, there does not exist a node x in $T_I(H)$ such that $|I(x)| = p$ and $common(x) > p(1 + 1/\ln(p))$ where $|I(x)|$ is the number of clauses in $I(x)$.

We will use observation 1 to show that if $n/r(n)$ doesn't grow too fast then, with probability tending to 1, $common(x)$ is small for every node x such that the number of distinct clause labels below x is $O(n^{\epsilon(n)})$, where $\epsilon(n)$ is not very small asymptotically. The following theorem and corollary state this more precisely.

Theorem 1:

Let I be an instance of k-SAT generated according to $M(n, r, k)$. Let $\omega(n)$ be any function that decreases asymptotically to 0 and is such that $\lim_{n\to\infty} n^{\omega(n)} = \infty$. Suppose that $n/r(n) = \lambda(n)$ obeys $\lambda(n) = o(n^{\omega(n)})$ and $\lambda(n) > \ln(2)/(-\ln(1 - 2^{-k}))$ for all $n > 0$. Let $\epsilon(n) = 1/(\ln(e^2 k^{k+2} \lambda(n)) + 3)$. Then the probability that there exists a subset I' of I with $p < n^{\epsilon(n)}$ clauses and such that $Icomm(I')$ is greater than $p(1 + 1/\ln(p))$ tends to 0 as n tends to infinity.

Proof:

In the rest of this proof and in the remaining proofs and discussion in this paper we use ϵ for $\epsilon(n)$ and λ for $\lambda(n)$ to avoid clutter. The probability that there exists a subset I' of I containing p clauses such that $Icomm(I') \geq a$ is less than the average number of such subsets. The average number of such subsets is the sum of the probabilities that each p clause subset I' of I has $Icomm(I') \geq a$. This is $\binom{n}{p}$ times the probability that $Icomm(I') \geq a$ where I' is a random p-clause subset of I. The probability that $Icomm(I') = i$ is the number of ways to construct I' such that $Icomm(I') = i$ divided by $\left(2^k\binom{r}{k}\right)^p$, the number of possible p-clause subsets of I. The number of ways to construct I' such that $Icomm(I') = i$ is less than the number of ways to construct I' such that $Icomm(I') = i$ if clauses were allowed to have duplicate or complementary literals. But the number of ways to construct I' such that $Icomm(I') = i$ and I' is allowed to have duplicate or complementary literals in the same clause is 2^{kp}, the number of ways to assign positive and negative literal values to kp literals, times the number of ways to partition kp literal place-holders into $kp - i$ variable groups with labels taken from 1 to r. Using braces to denote Stirling numbers of the second kind, the latter number is $\left\{{kp \atop kp-i}\right\}\binom{r}{kp-i}(kp-i)!$ (see [18], pages 133 and 134 for a detailed explanation of this quantity). It can easily be shown that $\binom{r}{k} \geq (r/k)^k$ for all integers $r \geq k > 0$. Furthermore, from the appendix, $\left\{{kp \atop kp-i}\right\} \leq (kp)^{2i}/i!$. Then, the average number of subsets of I containing p clauses and such that $Icomm(I') \geq a$ is less than

$$\begin{aligned}
&\binom{n}{p}\sum_{i=a}^{kp}\frac{\left\{{kp \atop kp-i}\right\}\binom{r}{kp-i}(kp-i)!}{(r/k)^{kp}}\\
&\leq \binom{n}{p}\sum_{i=a}^{kp}\frac{\left\{{kp \atop kp-i}\right\}r^{kp-i}}{(r/k)^{kp}}\\
&= \binom{n}{p}\sum_{i=a}^{kp}\frac{\left\{{kp \atop kp-i}\right\}k^{kp}}{r^i}\\
&\leq \binom{n}{p}\sum_{i=a}^{kp}\frac{(kp)^{2i}k^{kp}}{i!r^i}. \qquad (1)
\end{aligned}$$

Defining a new variable $j = i - p$, and bringing $\binom{n}{p}$ into the summand, we see that (1) is equal to

$$\sum_{j=a-p}^{kp-p}\frac{(kp)^{2(j+p)}n!k^{kp}}{(j+p)!p!(n-p)!r^{(j+p)}}$$

$$\leq \sum_{j=a-p}^{kp-p}\frac{\sqrt{n}(kp)^{2(j+p)}(n/e)^n e^{1/12n}k^{kp}}{2\pi\sqrt{(j+p)p(n-p)}((j+p)/e)^{(j+p)}(p/e)^p((n-p)/e)^{(n-p)}r^{(j+p)}} \qquad (2)$$

by Stirling's formula for factorials (that is, $x! = \sqrt{2\pi x}(x/e)^x e^{\theta/12x}, x > 0, 0 < \theta < 1$). Rearranging terms and noticing that $e^{1/12n}/\pi < 1$ since $n \geq 2$, (2) can be bounded from above by

$$\sum_{j=a-p}^{kp-p} \left(\frac{n}{r}\right)^p \left(\frac{n}{n-p}\right)^{(n-p)} \frac{(k\sqrt{e}p)^{2(j+p)}k^{kp}}{r^j p^p (j+p)^{(j+p)}} \sqrt{\frac{n}{4(j+p)p(n-p)}}$$

$$\leq \sum_{j=a-p}^{kp-p} \left(\frac{n}{r}\right)^p \left(\frac{n}{n-p}\right)^{(n-p)} \frac{(k\sqrt{e}p)^{2(j+p)}k^{kp}}{r^j p^p (j+p)^{(j+p)}}$$

since $n \geq 2$, $\epsilon < 1/3$ (and therefore $p < n^{1/3}$), and $p > 1$ implies

$$\sqrt{\frac{n}{4(j+p)p(n-p)}} \leq \sqrt{\frac{1}{4(j+p)p(1-n^{-2/3})}} \leq \frac{1}{1.2\sqrt{(j+p)p}} < 1$$

(actually we could derive a similar upper bound for $\epsilon < 1$ but it is unnecessary to do so to get our main result). By making use of the fact that $(1-p/n)^{(n-p)} \geq e^{-p}$ if n and p are positive and using $n/r = \lambda$ we can make the last sum

$$\leq \sum_{j=a-p}^{\infty} \frac{(e^2 k^{k+2}\lambda)^p (ek^2)^j p^p p^{2j}}{r^j (j+p)^p (j+p)^j}. \tag{3}$$

Suppose $a = p + p/\ln(p)$ (that is, $j \geq p/\ln(p)$). Then for sufficiently large n

$$\frac{ek^2p^2}{r(j+p)} < \frac{ek^2p}{r} < 1/2$$

since $\lambda = o(n^{\omega(n)})$ and $p < n^\epsilon$ where $\epsilon < 1/3$. Furthermore,

$$\frac{(e^2k^{k+2}p\lambda)^p}{(j+p)^p} < (e^2k^{k+2}\lambda)^p = (e^2k^{k+2}\lambda)^{\ln(p)(p/\ln(p))}.$$

Therefore, the summand of (3) is less than

$$\left(\frac{1}{2}\right)^{(j-p/\ln(p))} \left(\frac{p^{\ln(e^2k^{k+2}\lambda)}ek^2p}{r}\right)^{p/\ln(p)}$$

for sufficiently large n. Hence the sum (1) is less than

$$2\left(\frac{p^{\ln(e^2k^{k+2}\lambda)}ek^2p}{r}\right)^{p/\ln(p)}$$

for sufficiently large n if $a = p(1+1/\ln(p))$. Thus, the probability that there exists a subset I' of I containing n^ϵ or fewer clauses such that $Icomm(I') > p(1+1/\ln(p))$ is less than

$$\sum_{p=2}^{n^\epsilon} 2\left(\frac{p^{\ln(e^2k^{k+2}\lambda)}ek^2p}{r}\right)^{p/\ln(p)}. \tag{4}$$

The derivative of the summand of (4) with respect to p is

$$2\left(\ln(e^2k^{k+2}\lambda)+1+\frac{\ln(ek^2/r)}{\ln(p)}\left(1-\frac{1}{\ln(p)}\right)\right)\left(\frac{p^{(\ln(e^2k^{k+2}\lambda)+1)}ek^2}{r}\right)^{p/\ln(p)}.$$

For sufficiently large r, $\ln(ek^2/r)$ is a negative number. Hence, for sufficiently large r, $\ln(ek^2/r)(1-1/\ln(p))/\ln(p)$ becomes more positive as p increases when $p > 2$. Furthermore,

$$\left(\frac{p^{(\ln(e^2k^{k+2}\lambda)+1)}ek^2}{r}\right)^{p/\ln(p)}$$

increases as p increases, $p > 1$. Therefore, the derivative of the summand of (4) is monotonically increasing with p, $p > 2$, and is maximum at either $p = 2$, $p = 3$ or $p = n^\epsilon$. At $p = 2$, it is straightforward to check that for any $\epsilon < 1/3$

$$2\left(\frac{2^{(\ln(e^2k^{k+2}\lambda)+1)}ek^2}{r}\right)^{2/\ln(2)} < n^{-2\epsilon}$$

for large n if $\lambda = o(n^{\omega(n)})$ and k is fixed. Similarly for $p = 3$. At $p = n^\epsilon$, where $\epsilon = 1/(\ln(e^2k^{k+2}\lambda)+3)$, we have

$$\begin{aligned}
&2\left(\frac{n^{(\ln(e^2k^{k+2}\lambda)+1)\epsilon}ek^2}{r}\right)^{n^\epsilon/\epsilon\ln(n)}\\
&\qquad=2\left(n^{(\ln(e^2k^{k+2}\lambda)+1)\epsilon-1}ek^2\lambda\right)^{n^\epsilon/\epsilon\ln(n)}\\
&\qquad\leq 2\left(n^{-2\epsilon}ek^2\lambda\right)^{n^\epsilon/\epsilon\ln(n)} < n^{-2\epsilon}
\end{aligned}$$

for large enough n if $\lambda = o(n^{\omega(n)})$. Since the summand of (4) has a maximum less than $n^{-2\epsilon}$, the sum (4) is less than $n^{-\epsilon}$. But $1/\ln(n^\epsilon) = (\omega(n)\ln(n)+\ln(e^2k^{k+2})+3)/\ln(n)$ which tends to zero as n gets large so $n^{-\epsilon}$ tends to zero. This proves the theorem.

Corollary 1:

Let I be an instance of k-SAT generated according to $M(n,r,k)$. Let $\omega(n)$ be any function that decreases asymptotically to 0 and is such that $\lim_{n\to\infty} n^{\omega(n)} = \infty$. Suppose that $n/r(n) = \lambda(n)$ obeys $\lambda(n) = o(n^{\omega(n)})$ and $\lambda(n) > \ln(2)/(-\ln(1-2^{-k}))$ for all $n > 0$. Then the following statement is true with probability tending to 1. For all H and all nodes x in $T_I(H)$ such that the number of distinct clause labels below x is $p < n^{1/(\ln(e^2k^{k+2}\lambda(n))+3)}$, $common(x) \leq p(1+1/\ln(p))$.

Proof:

Follows from Theorem 1, observation 1 and the fact that almost all instances are unsatisfiable if $n/r > \ln(2)/(-\ln(1-2^{-k}))$.

For the sake of simplicity we drop the subscript from $T_I(H)$ in what follows. Corollary 1 gives a property that any search rearrangement backtrack tree has with probability tending to 1 if instances are generated according to $M(n,r,k)$. In Lemma 1 we show that $h(x) \geq kp(x) - 2 * common(x)$ for all nodes x in $T(H)$. This and the fact that $k \geq 3$ means that, with probability tending to 1, $h(x) \geq p(x)(1 - 2/\ln(p))$ for any node x that is the root of a subtree containing $p(x) < n^{\epsilon}$, $\epsilon = 1/(\ln(e^2 k^{k+2}\lambda) + 3)$, distinct clause labels. In Lemma 2 we derive an important property shared by almost all random graphs we consider. The property is that no variable appears in more than $(\ln(n)+1)k\lambda$ clauses. In Lemma 3 this property is used to show that, with probability tending to 1, $h(x) \geq h(y) - (\ln(n)+1)k\lambda$ where x is a node in $T(H)$ which is the parent of y (that is, the h function cannot decrease by more than $(\ln(n)+1)k\lambda$ per node as we move toward the root of $T(H)$). So, with probability tending to 1, for any node x in $T(H)$ such that $h(x) > L$, the number of nodes on the path from x to the root of $T(H)$ is at least $L/(\ln(n)+1)k\lambda$ (Theorem 3). In Theorem 4 we show that, with probability tending to 1, there exists a node x such that $n^{\epsilon/2} > p(x) \geq n^{\epsilon/2}/2$. This means there is at least one node x in $T(H)$ such that $h(x) > n^{\epsilon/2}(1 - 4/\ln(n^{\epsilon}))$ and that the number of nodes on the path from that node to the root is $n^{\epsilon/2}(1-4/\ln(n^{\epsilon}))/(\ln(n)+1)k\lambda$. We slice off all nodes in $T(H)$ that are deeper than $n^{\epsilon/2}(1-4/\ln(n^{\epsilon}))/(\ln(n)+1)k\lambda$ and call each node at that depth a bottomnode. In Theorem 5 we show that, with probability tending to 1, on the path from all bottomnodes to the root there are at least $2n^{\epsilon/4}(1-4/\ln(n^{\epsilon}))/(\ln(n)+1)k\lambda$ nodes for which both children have bottomnodes as descendants. This implies, with probability tending to 1, an exponential treesize for $T(H)$ if λ is fixed and superpolynomial treesize if $o(n^{1/\ln\ln(n)}) = \lambda(n)$ and $\lim_{n\to\infty} \lambda(n) = \infty$ (Theorems 6, 7 and 8).

First we derive some relationships between $h(x)$, $common(x)$ and $var(x)$.

Theorem 2:

For all H and x in $T(H)$, $h(x) = kp(x) - common(x) - var(x)$.

Proof:

Recall that $common(x) + var(x)$ is the number of distinct edge labels at or below x. The rest follows from the definition of $h(x)$.

Theorem 2 leads to the following useful relationship between $h(x)$ and $common(x)$.

Lemma 1:

For any H function and x in $T(H)$, $h(x) \geq kp(x) - 2 * common(x)$.

Proof:

Since every variable below x is associated with at least two distinct edge labels, $c(x) \geq 2 * var(x)$. Therefore, $var(x) \leq common(x)$. This and Theorem 2 imply $h(x) \geq kp(x) - 2 * common(x)$.

The next two lemmas and theorem show that, for any H function and any node x in $T(H)$ such that $h(x) \geq n^{\epsilon}$, the length of the path from root of $T(H)$ to x is great, with probability tending to 1, if $\lambda = o(n^{\omega(n)})$ and $\lambda(n) > \ln(2)/(-\ln(1-2^{-k}))$ for all $n > 0$.

Lemma 2:

Let $\omega(n)$ be any function of n that tends to 0 asymptotically and is such that $\lim_{n\to\infty} n^{\omega(n)} = \infty$. The probability that some variable appears in more than $(\ln(n)+1)k\lambda(n)$ clauses of an instance of k-SAT generated according to $M(n,r,k)$ tends to 0 as n tends to infinity if $\lambda(n) = o(n^{\omega(n)})$ and $\lambda(n) > \ln(2)/(-\ln(1-2^{-k}))$ for all $n > 0$.

Proof:

Let v be a variable taken from V. The average number of clauses containing v is $kn/r = k\lambda$. The probability that v is in at least $(\ln(n)+1)k\lambda$ clauses is

$$\sum_{i=(\ln(n)+1)k\lambda}^{n} \binom{n}{i}\left(\frac{k}{r}\right)^{i}\left(1-\frac{k}{r}\right)^{n-i} \leq e^{-\ln^2(n)k\lambda/3}$$

from the Chernoff bound for binomial distributions [1] and [8]. The average number of variables that appear in at least $(\ln(n)+1)k\lambda$ clauses is therefore

$$re^{-\ln^2(n)k\lambda/3} = \frac{r}{e^{\ln(n)\ln(n)k\lambda/3}} = \frac{r}{n^{\ln(n)k\lambda/3}}$$

$$= \frac{1}{\lambda n^{\ln(n)k\lambda/3-1}} \to 0 \text{ as } n \to \infty.$$

Since the average number of variables that appear in at least $(\ln(n)+1)k\lambda$ clauses is an upper bound on the probability that there exists a variable that appears in at least $(\ln(n)+1)k\lambda$ clauses the lemma is proved.

In what follows we show that the search tree for any H must be exponentially large if the input has the properties stated in Lemma 2 and Corollary 1.

Lemma 3:

Let $\omega(n)$ be any function that tends to zero asymptotically and is such that $\lim_{n\to\infty} n^{\omega(n)} = \infty$. The following statement is true with probability tending to 1. For all H functions and parent nodes x in $T(H)$ with child y, $h(x) \geq h(y) - (\ln(n)+1)k\lambda(n)$ if $\lambda(n) = o(n^{\omega(n)})$ and $\lambda(n) > \ln(2)/(-\ln(1-2^{-k}))$ for all $n > 0$.

Proof:

Let there be s_y labels associated with the edge connecting x with y and s_z labels associated with the edge connecting x to z. Let $N_{yz}(i)$ denote the number of clauses that appear as edge labels i times in the path from y to a leaf of $T(H)$ labeled by such clauses and in the path from z to a leaf of $T(H)$ labeled by such clauses (note that the labels associated with these clauses contribute $k-i$ to $h(x)$ but twice this to $h(y)+h(z)$). Therefore,

$$h(x) = h(y) + h(z) - (s_y + s_z) - \sum_{i=1}^{k}(k-i)N_{yz}(i). \qquad (5)$$

Observe that in equation (5) $h(z) - \sum_{i=1}^{k}(k-i)N_{yz}(i) \geq 0$. Hence $h(x) \geq h(y) - (s_y + s_z)$. But, from Lemma 2, the probability that no variable appears in more than $(\ln(n)+1)k\lambda$ clauses tends to 1. Since the variable associated with x is in clauses with labels associated with edges connecting x to its children, we have that $s_y + s_z \leq (\ln(n)+1)k\lambda$ for all H and x in $T(H)$ with probability tending to 1. The lemma follows.

Theorem 3:

Let $L(n)$ be any positive integer function of n and let $\omega(n)$ be any function of n that tends to 0 asymptotically and is such that $\lim_{n\to\infty} n^{\omega} = \infty$. The following statement holds with probability tending to 1. For all H functions, the pathlength of any path from the root of $T(H)$ to a node x such that $h(x) \geq L(n)$ is at least $L(n)/(\ln(n)+1)k\lambda$ if $\lambda(n) = o(n^{\omega(n)})$ and $\lambda(n) > \ln(2)/(-\ln(1-2^{-k}))$ for all $n > 0$.

Proof:

Follows immediately from Lemma 3 and the fact that $h(\text{root}(T(H))) = 0$.

Theorem 4:

Let $\omega(n)$ be any function of n that tends to 0 asymptotically and is such that $\lim_{n\to\infty} n^{\omega(n)} = \infty$. Let $0 < \gamma \leq 1$ be fixed and let $\epsilon(n) = 1/(\ln(e^2k^{k+2}\lambda(n)) + 3)$. The following statement is true with probability tending to 1. For all H functions and $k \geq 3$, there is at least one subtree of $T(H)$ with at least $n^{\gamma\epsilon(n)}/2$ and at most $n^{\gamma\epsilon(n)}$ distinct clause labels below its root if $\lambda(n) = o(n^{\omega(n)})$ and $\lambda(n) > \ln(2)/(-\ln(1-2^{-k}))$ for all $n > 0$.

Proof:

From the root of $T(H)$ move toward a leaf y visiting nodes as follows: at each visited node x, visit next the child of x which has the greatest number of distinctly labeled leaves beneath it (decide ties arbitrarily). Call the path just traced P. Let x be a node on P. The number of distinctly labeled leaves beneath the parent of x is no greater than twice $p(x)$ because the number of distinctly labeled leaves beneath the sibling of x is less than $p(x)$ (otherwise we would have moved in the direction of the sibling on the way down). Furthermore, the number of distinctly labeled leaves beneath the parent of x is at least $p(x)+1$ since the clause labeling the sibling of x cannot be below x. Thus, if we move up s nodes from y we will be at a node which has at least $s+1$ and at most 2^s distinctly labeled leaves beneath it. We can certainly move up P from y as long as the number of distinct clauses beneath the currently visited node is less than 8 since at least 8 clauses are required for a refutation of k-SAT where $k \geq 3$. From Lemma 1 and Corollary 1 we have that, for any node x on P such that $p(x) < n^{\epsilon}$, $h(x) > p(x)(1 - 2/\ln(p(x))$, with probability tending to 1. Thus, if $8 \leq p(x) < n^{\gamma\epsilon}$, $0 < \gamma \leq 1$, then $h(x) > 0$ hence x is not the root of $T(H)$. Therefore, we can move up P from y to the last node z such that $p(z) \geq n^{\gamma\epsilon}/2$. Since $p(\text{father}(z))$ can be at most double $p(z)$ we have $p(z) < n^{\gamma\epsilon}$. The node z is the one required to prove the theorem.

We call a node that is the root of a subtree of $T(H)$ containing between $n^{\epsilon/2}/2$ and $n^{\epsilon/2}$, $\epsilon = 1/(\ln(e^2 k^{k+2}\lambda) + 3)$, distinct clause labels a bignode. Observe that within the proof of Theorem 4 it was shown that if x is a bignode then, since $k \geq 3$, and $n^{\epsilon/2}/2 \leq p(x) < n^{\epsilon/2}$, $h(x) \geq n^{\epsilon/2}(1 - 4/(\epsilon\ln(n) - 2\ln(2)))/2$. Therefore, Theorems 3 and 4 say that, with probability tending to 1, bignodes exist for every H function and that all paths from the root of $T(H)$ to bignodes must contain at least $n^{\epsilon/2}(1 - 4/(\epsilon\ln(n) - 2\ln(2)))/2(\ln(n)+1)k\lambda$ nodes. Call a node at depth $n^{\epsilon/2}(1 - 4/(\epsilon\ln(n) - 2\ln(2)))/2(\ln(n)+1)k\lambda$ a bottomnode. All bignodes must be descendents of bottomnodes. Hence at least one bottomnode exists in $T(H)$. The next two theorems tell us that, for any H function, the number of bottomnodes in $T(H)$ is exponential with probability tending to 1 if λ is fixed and $\lambda > \ln(2)/(-\ln(1-2^{-k}))$. Theorem 8 says that the number of bottomnodes is superpolynomial with probability tending to 1 if $\lambda = o(n^{1/\ln\ln(n)})$ and $\lambda > \ln(2)/(-\ln(1-2^{-k}))$ for all $n > 0$.

Theorem 5:

Let $\omega(n)$ be any function of n that tends to 0 asymptotically and is such that $\lim_{n\to\infty} n^{\omega(n)} = \infty$. Let $\epsilon(n) = 1/(\ln(e^2 k^{k+2}\lambda(n))+3)$. The following statement holds with probability tending to 1. If $\lambda(n) = o(n^{\omega(n)})$ and $\lambda(n) > \ln(2)/(-\ln(1-2^{-k}))$ for all $n > 0$ then, for all H functions, on every path from the root of $T(H)$ to a bottomnode there are at least

$$n^{\epsilon(n)/4}(1 - 4/\ln(n^{\epsilon(n)}))/(\ln(n)+1)k\lambda(n)$$

nodes x such that both children of x are ancestors of bottomnodes.

Proof:

The restriction on λ causes random instances of k-SAT to be unsatisfiable with probability tending to 1 so in what follows we can skip over the cases where the required trees don't exist and consider only those trees which are refutation trees (that is, the trees in which all leaves are labeled with clause labels). Consider any path P from root to bottomnode in $T(H)$. For some nodes on P both children are ancestors of bottomnodes and for the remaining nodes on P exactly one child is an ancestor of a bottomnode (we call the other child an Orphan and its subtree an Orphaned subtree). Call nodes of the first kind Binary and nodes of the second kind Unary. The Capital letters distinguish Binary and Unary nodes from ordinary binary and unary nodes of a search tree. We use the terms Binary and Unary because Binary nodes have two connections to subtrees containing bottomnodes and Unary nodes have only one. It will be understood in what follows that Binary and Unary nodes are on P and that Orphans and Orphaned subtrees are attached to Unary nodes. Let P_P denote the number of distinct clauses labeling the leaves of Orphaned subtrees.

Suppose that $P_P < n^{\epsilon/4}$. All of the clause labels associated with each edge connecting an Orphan node with a Unary are different from the clause labels associated with all other edges connecting Orphan nodes to Unary nodes since a clause label associated with any edge cannot appear below the sibling of its endpoint. But $P_P < n^{\epsilon/4}$ so the number of Unary nodes is less than $n^{\epsilon/4}$. Therefore, there can be no more than $n^{\epsilon/4}$ Unary nodes on the path from root to bottomnode. Since the number of Binary nodes is the number of nodes on P minus the number of Unary nodes, and since the number of nodes on P is $n^{\epsilon/2}(1-(4/(\ln(n^{\epsilon})-2\ln(2)))/2(\ln(n)+1)k\lambda$, the number of Binary nodes on P must be at least

$$\frac{n^{\epsilon/2}(1-4/(\ln(n^{\epsilon})-2\ln(2)))}{2(\ln(n)+1)k\lambda} - n^{\epsilon/4} > \frac{n^{\epsilon/4}(1-4/\ln(n^{\epsilon}))}{(\ln(n)+1)k\lambda}$$

for sufficiently large n, and the theorem holds.

Now suppose that $P_P \geq n^{\epsilon/4}$. We know that no Orphaned subtree contains a bignode. Each Orphaned subtree cannot contain more than $n^{\epsilon/2}$ distinct clauses since otherwise we could trace a path through the subtree, as in Theorem 4, and get to a bignode. Then $P_P < n^{\epsilon}$ (an upper bound on the product of the pathlength of P and the maximum number of distinct clauses below each Orphan of P). Let $I(P)$ be the set of distinct clauses labeling leaves of Orphaned subtrees. Let B_P be the set of edges on P which connect a Binary node to its child on P. Each literal in $I(P)$ corresponds to a distinct edge label in Orphaned subtrees, edges connecting Unary nodes to their children, and edges in B_P. Let h_P denote the number of distinct edge labels which are associated *only* with edges in B_P and are due to leaves of all Orphaned subtrees. Recall that $T(H)$ is a refutation tree so P_P distinct clauses labeling leaves of $T(H)$ generate kP_P distinct edge labels in $T(H)$. Define $L_P = kP_P - h_P$. That is, L_P is the number of distinct edge labels which are associated with edges connecting Unary nodes to their children and edges within Orphaned subtrees and are due to leaves of all Orphaned subtrees. Let V_P denote the set of variables which are associated with Unary nodes and nodes in Orphaned subtrees. See Figure 2 for an example showing sets mentioned above. Figure 2 also illustrates sets mentioned below.

In this paragraph we show that $Icomm(I(P)) \geq L_P - |V_P|$. Let VBD_P denote the set of all variables that appear at least two times in $I(P)$ but are not in V_P. These variables are associated *only* with Binary nodes. Let LUS_P denote the set of edge labels which are in edges that connect Unary nodes to their children and are associated with variables that appear exactly once in $I(P)$. Since there is one edge label in LUS_P for every variable that is both associated with a Unary node and in $I(P)$ exactly once, $|V_P| + |VBD_P| - |LUS_P|$ is the number of variables in $I(P)$ which are in at least two clauses of $I(P)$. Let $LOUD_P$ denote the set of edge labels which are in Orphaned subtrees along P or are associated with edges that connect Unary nodes with their children on P and are associated with variables that appear two or more times in $I(P)$. Let LBD_P denote the set of edge labels not in $LOUD_P$ which label edges in B_P and are associated with variables that appear two or more times in $I(P)$. Note that $|LBD_P| \geq |VBD_P|$ since there is at least one edge label associated with an edge incident on a variable in VBD_P which is not in $LOUD_P$. Recall that $Icomm(I(P))$ is the total number of literals in $I(P)$ that are associated with variables that appear two or more times in $I(P)$ minus the number of such variables. Then,

$$\begin{aligned} Icomm(I(P)) &= |LOUD_P| + |LBD_P| - |V_P| - |VBD_P| + |LUS_P| \\ &\geq |LOUD_P| - |V_P| + |LUS_P|. \end{aligned}$$

But $|LOUD_P| + |LUS_P| = L_P$ since $LOUD_P \cap LUS_P = \Phi$. It follows that $Icomm(I(P)) \geq L_P - |V_P|$.

Therefore, $h_P \geq kP_P - Icomm(I(P)) - |V_P|$.

In this paragraph we show that $|V_P| \leq P_P$. Create a forest T' from $T(H)$ by removing all nodes and edges from $T(H)$ except the Unaries, edges connecting Unaries to their children, and Orphaned subtrees along P. Construct a tree T'' from T' by appending each Unary to the free edge of another Unary so that all Unaries are in the same order as they were on P. Retain all edge label and variable associations that existed originally. The number of variables associated with nodes in T'' is, by definition, $|V_P|$ (exactly the Unary nodes and Orphaned subtrees remain). Perform a depth first search on T'' and mark leaves that contain labels distinct from all other previously marked leaves. Eliminate all nodes that are not ancestors of marked nodes, edges on paths to unmarked leaves and the unmarked leaves themselves. The result is a tree containing a number of binary and unary nodes (lower case binary and unary nodes are ordinary binary and unary nodes). Call the eliminated edges that were connected to unaries "missing" (so there is one missing edge for each unary in T''). Except for one Unary, Unaries on P can appear either as binaries in T'' or as unaries in T'' with a clause in $I(P)$ labeling the missing edge. The exception is the deepest Unary in P. This Unary is the *special unary* node in T'' and possibly has no clause in $I(P)$ labeling its missing edge. The number of leaves remaining in T'' is P_P, each one representing a distinct clause. Except for the special unary, a variable v associated with a unary node in T'' must be the same as the variable associated with a previously visited node in T''. This is because some clause must label the edge missing from the unary and the edge is missing because the clause has already been visited; but, since v is in the clause, a node asscociated with v must have been visited. Consequently, except for the variable associated with the special unary, every variable in T'' is associated with some binary in T''. The number of binary nodes in T'' is $P_P - 1$. Hence the number of variables in T'' (that is, $|V_P|$) is at most P_P (after adding 1 for the special unary).

Thus, $h_P \geq kP_P - Icomm(I(P)) - P_P$. We next apply our familiar bound on $Icomm(I(P))$.

In what follows we make statements which are true for all H applied to almost all instances of k-SAT generated according to $M(n, r, k)$ if $\lambda(n) = o(n^{\omega(n)})$ and $\lambda > \ln(2)/(-\ln(1-2^{-k}))$ for all n> 0; these conditions are omitted for brevity. Since $|I(P)| = P_P < n^{\epsilon}$ we can obtain from Theorem 1 that $Icomm(I(P)) < P_P(1 + 1/\ln(P_P))$. Therefore, since $k \geq 3$, $h_P \geq (k-1)P_P - P_P(1 + 1/\ln(P_P)) \geq P_P(1 - 1/\ln(P_P))$. Since $P_P > n^{\epsilon/4}$ we have that $h_P > n^{\epsilon/4}(1 - 4/\ln(n^{\epsilon}))$. The labels counted by h_P must be spread over edges in B_P. From Theorem 3, no edge in B_P can receive more than $(\ln(n)+1)k\lambda$ labels. Hence the number of edges in B_P must be at least $n^{\epsilon/4}(1-4/\ln(n^{\epsilon}))/(\ln(n)+1)k\lambda$. Since there is one edge in B_P for every Binary node on P, the number of Binary nodes must be at least $n^{\epsilon/4}(1-4/\ln(n^{\epsilon}))/(\ln(n)+1)k\lambda$ in this case. This completes the theorem.

Theorem 6:

Let $T(H)$ be a search tree generated by SRB for any H function. If on every path from the root of $T(H)$ to a bottomnode there are at least s nodes whose left child and right child are ancestors of bottomnodes then $T(H)$ must have at least 2^s nodes.

Proof:

Compress $T(H)$ by eliminating all nodes from $T(H)$ which are not bottomnodes or do not have two children which are ancestors of bottomnodes. The result is a binary tree of depth at least s. The number of nodes in such a tree is at least 2^s.

The following two theorems state the main results.

Theorem 7:

The following statement holds with probability tending to 1. For all H functions, SRB requires $O(2^{n^\alpha})$ time, $\alpha > 0$, under $M(n, r, k)$ for any fixed $\lambda > \ln(2)/(-\ln(1-2^{-k}))$.

Proof:

Follows from Theorem 5, Theorem 6 where s in Theorem 6 is

$$n^{\epsilon/4}(1-4/\ln(n^\epsilon))/(\ln(n)+1)k\lambda$$

and $\epsilon = 1/(\ln(\lambda e^2 k^{k+2})+3)$ and the fact that each node of $T(H)$ must be visited and requires at least one unit of time.

Theorem 8:

The following statement holds with probability tending to 1. For all H functions, SRB requires superpolynomial time under $M(n, r, k)$ for all functions $\lambda(n)$ satisfying $o(n^{1/\ln\ln(n)}) = \lambda(n)$, $\lim_{n\to\infty} \lambda(n) = \infty$.

Proof:

As in Theorem 7, the number of nodes in $T(H)$ is at least

$$2^{\left(\frac{1-4/\ln n^\epsilon}{(1+\ln(n))k\lambda}\right)n^{\epsilon/4}}$$

$$= 2^{\left(\frac{1-(4\ln(\lambda)+\Theta(1))/\ln(n)}{k\ln(n)\lambda+\Theta(k\lambda)}\right)n^{1/(4\ln(\lambda)+\Theta(1))}}$$

$$= 2^{\left(\frac{1-\Theta(\ln(\lambda))/\ln(n)}{k\ln(n)\lambda}\right)n^{(4\ln(\lambda))^{-1}(1-\Theta((4\ln(\lambda))^{-1}))}}$$

But $\Theta(\ln(\lambda))/\ln n = o(1/\ln\ln n)$ so the last term is

$$2^{\Theta((\ln(n)\lambda)^{-1})n^{\Theta((\ln(\lambda))^{-1})}}.$$

But $\lambda(n) = o(n^{1/\ln\ln n})$ so the last term is

$$2^{\Theta((\ln(n)\lambda)^{-1})n^{1/o(\ln\ln(n)/\ln(n))}} = 2^{1/o((\ln(n))^{-1})}$$

which grows too fast to be polynomial.

According to Theorem 8 superpolynomial time is achieved for a rather large range of relationships of n to r. For example, we get superpolynomial time, almost always, if $n/r(n) = (\ln(n))^\beta$ for any constant β. We have shown superpolynomial time, almost always, for $n/r(n)$ almost as high as $n^{1/\ln\ln(n)}$ which is very nearly n to a constant power.

3. Davis-Putnam Procedure

As stated in the introduction, the Davis-Putnam Procedure contains three principal components: decomposing I into two subinstances I_1 and I_2, the unit-clause rule and the pure-literal rule. It is the pure-literal rule that appears to be preventing the analysis from carrying over. However we can show that for the Davis-Putnam Procedure with any heuristic function there is a Search Rearrangement Backtracking algorithm which expands fewer nodes of the search tree when inputs are unsatisfiable. This means that our result holds also for the Davis-Putnam Procedure.

To see this, first develop a search tree for a given Davis-Putnam Procedure and input I which is patterned after the search tree developed for SRB (that is, nodes are associated with variables, leaves are labeled with clause names, edges are associated with leaf labels, etc.). Observe any node x in the tree which corresponds to a point in the application of the Davis-Putnam Procedure where the pure-literal rule is used for the last time before backing up. That node has only one child which we denote by y. Create another child z of x and subtree under z which is exactly the same as the subtree under y. Because the pure-literal rule is applied at x, the edge $\langle x, z\rangle$ has no label associated with it. Therefore, we can replace the subtree rooted at x with the subtree rooted at z and still have a search tree corresponding to a verification of unsatisfiability; the difference is that there is one less application of the pure-literal rule and the tree has fewer nodes. Continuing this until all pure-literal applications are removed results in a search tree corresponding to a Search Rearrangement Backtracking algorithm applied to I. This tree is smaller than the one corresponding to the application of the Davis-Putnam Procedure on I and has the property that there is at least one clause label associated with every edge. Therefore, the results of the last section apply to any form of Davis-Putnam Procedure.

4. Conclusions

We have presented an analysis which shows that any form of Search Rearrangement Backtracking requires $O(2^{n^\alpha})$ time, $\alpha > 0$, for almost all instances of k-SAT generated according to $M(n,r,k)$ if $n/r = \lambda$ where λ is fixed and is such that almost all instances are unsatisfiable. We have also shown superpolynomial running time for almost all instances of k-SAT if $\lambda(n) = o(n^{1/\ln\ln(n)})$ and $\lim_{n\to\infty} \lambda(n) = \infty$. The proof of this is interesting because it is based on a *structural* property of instances of k-SAT. We have shown that these results also apply to any form of the Davis-Putnam Procedure.

5. Acknowledgement

I wish to thank Paul W. Purdom for his careful reading of this manuscript and his valuable suggestions regarding the relevance, clarity and presentation of its contents.

6. References

[1] Angluin, D. and Valiant, L., "Fast probabilistic algorithms for hamiltonian circuits and matchings," *JCSS* **18** (1979), pp. 155-193.

[2] Bitner, J. R. and Reingold, E. M., "Backtrack programming Techniques," *CACM* **18**, No. 11 (1975), pp. 651-656.

[3] Brown, C. A. and Purdom, P. W., "An average time analysis of backtracking," *SIAM J. Comput.* **10** (1981), pp. 583-593.

[4] Chao, M. T. and Franco, J., "Probabilistic analysis of two heuristics for the 3-Satisfiability problem," *SIAM J. Comput.* **15** (1986), pp. 1106-1118.

[5] Chao, M. T. and Franco, J., "Probabilistic analysis of a generalization of the unit clause rule for the Satisfiability problem," Tech. Report No. 165, Indiana University (1985).

[6] Davis, M. and Putnam, H., "A computing procedure for quantification theory," *J.ACM* **7** (1960), pp. 201-215.

[7] Davis, M., Putnam, H. and Loveland, D., "A machine program for theorem proving," *CACM* **5** (1962), pp. 394-397.

[8] Erdos, P. and Spencer, J., *Probabilistic Methods in Combinatorics*, Academic Press, 1974.

[9] Franco, J., "On the probabilistic performance of algorithms for the Satisfiability problem," *Information Processing Letters* **23** (1986), pp. 103-106.

[10] Franco, J. and Paull, M., "Probabilistic analysis of the Davis Putnam Procedure for solving the satisfiability problem," *Discrete Applied Mathematics* **5** (1983), pp. 77-87.

[11] Franco, J., Plotkin, J. M. and Rosenthall, J. W., "Correction to probabilistic analysis of the Davis-Putnam procedure for solving the satisfiability problem," *Discrete Applied Mathematics* **17** (1987), pp. 295-299.

[12] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco, 1979.

[13] Goldberg, A., Purdom, P. W. and Brown, C. A., "Average time analysis of simplified Davis-Putnam procedures," *Information Processing Letters* **15** (1982), pp. 72-75.

[14] Purdom, P. W., "Search Rearrangement Backtracking and polynomial average time," *Artificial Intelligence* **21** (1983), pp. 117-133.

[15] Purdom, P. W. and Brown, C. A., "The pure literal rule and polynomial average time," *SIAM J. Comput.* **14** (1985), pp. 943-953.

[16] Purdom, P. W., Brown, C. A., "An analysis of backtracking with search rearrangement," *SIAM J. Comput.* **12** (1983), pp. 717-733.

[17] Purdom, P. W., Brown, C. A. and Robertson, E. L., "Backtracking with multi-level dynamic search rearrangement," *Acta Informatica* **15** (1981), pp. 99-113.

[18] Purdom, P. W. and Brown, C. A., *The Analysis of Algorithms*, Holt, Rinehart and Winston, 1985.

APPENDIX

Lemma:

$\left\{ {n \atop n-x} \right\} \leq \frac{n^{2x}}{x!}$ where {} denotes stirling numbers of the second kind.

Proof:

By induction on n.

Basis: $\left\{ {1 \atop 1} \right\} = 1 = 1^0/0!$.

Induction Step: From the definition of stirling numbers of the second kind

$$\left\{ {n \atop n-x} \right\} = (n-x)\left\{ {n-1 \atop n-x} \right\} + \left\{ {n-1 \atop n-x-1} \right\}$$

$$\leq (n-x)\frac{(n-1)^{2(x-1)}}{(x-1)!} + \frac{(n-1)^{2x}}{x!} \quad \text{by hypothesis}$$

$$= \frac{n^{2x}}{x!}\left[\frac{x(n-1)^{2x-2}(n-x)}{n^{2x}} + \left(\frac{n-1}{n}\right)^{2x}\right]$$

$$= \frac{n^{2x}}{x!}\left[\left(\frac{n-1}{n}\right)^{2x}\left(\frac{x}{n-1}\left(1-\frac{x-1}{n-1}\right)+1\right)\right].$$

The lemma holds if the term within brackets is less than or equal to 1. We show this as follows. Rewrite $\left(\frac{n-1}{n}\right)^{2x}$ as $e^{2x\ln(1-1/n)}$. Notice that

$$\frac{x}{n-1}\left(1-\frac{x-1}{n-1}\right)+1 < 1+\frac{x}{n-1}$$

$$< \sum_{i=0}^{x}\binom{x}{i}\left(\frac{1}{n-1}\right)^{i} = \left(1+\frac{1}{n-1}\right)^{x} = e^{-x\ln(1-1/n)}.$$

Therefore, the term in brackets is less than $e^{2x\ln(1-1/n)} * e^{-x\ln(1-1/n)} = e^{x\ln(1-1/n)} <$ 1.

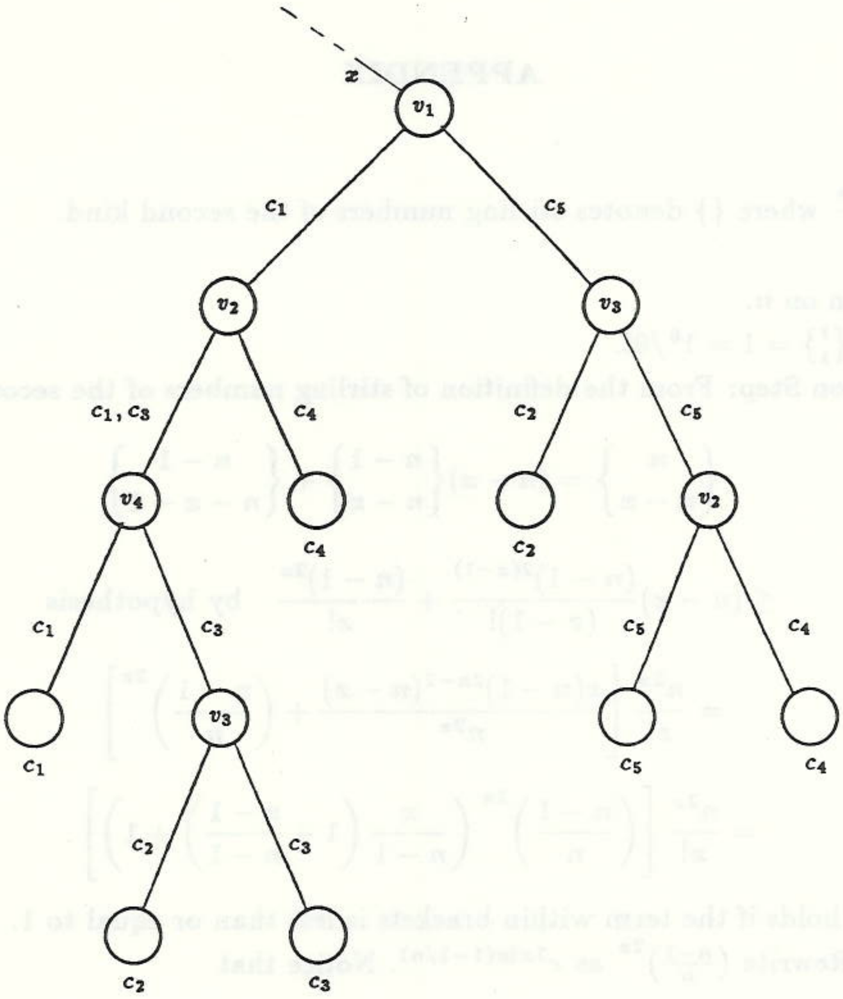
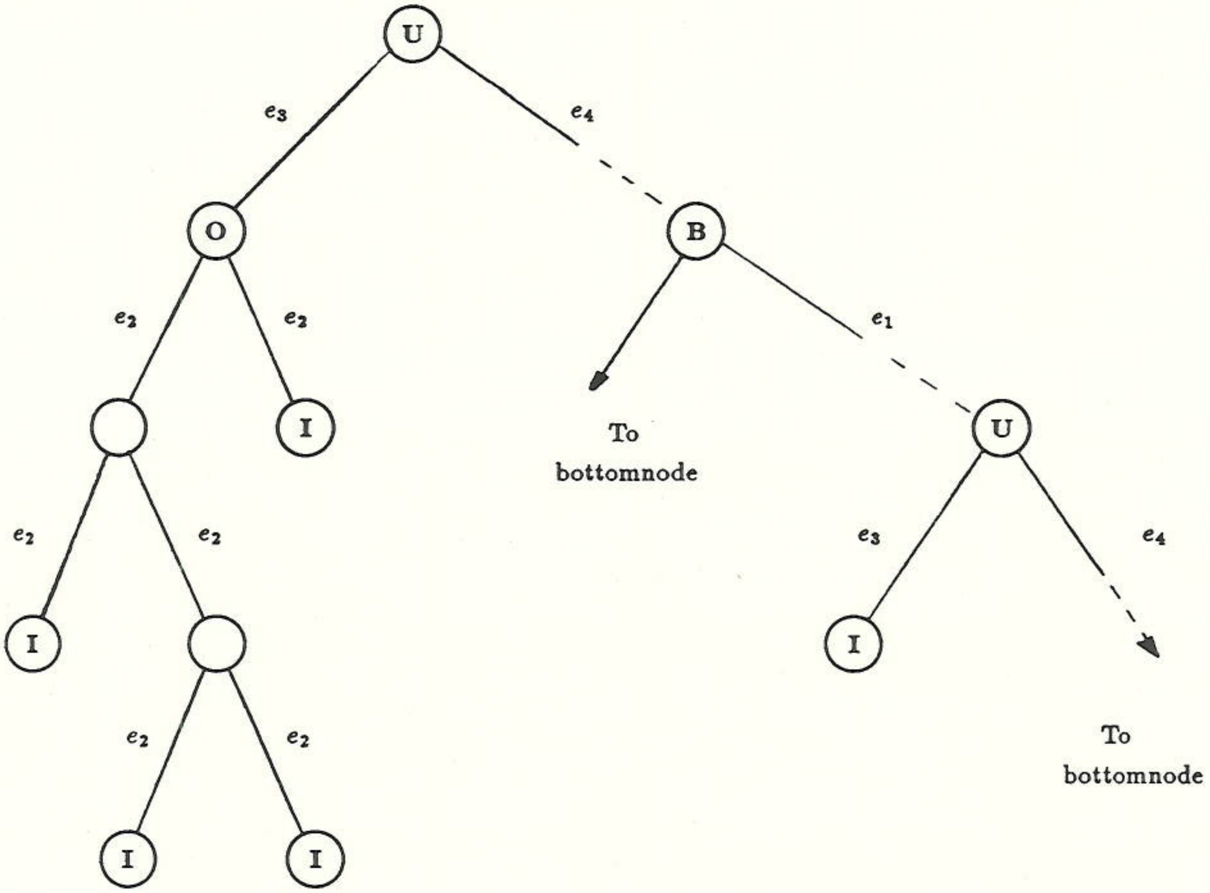


Figure 1: A subtree of T_I rooted at x showing leaf labels beneath the leaves and clause labels associated with edges (literals are implied). Variables associated with nodes are shown inside the nodes. For this subtree $I(x) = (\bar{v}_1, \bar{v}_2, \bar{v}_4)(\bar{v}_3, v_5, v_6)(\bar{v}_2, v_3, v_4)(v_2, v_5, v_6)(v_1, \bar{v}_2, v_3)$, $common(x) = 7$ and $h(x) = 3 * 5 - 11 = 4$.



Binaries : These nodes are marked **B**

Unaries : These nodes are marked **U**

Orphans : These nodes are marked **O**

$\mathbf{V_P}$: These variables are associated with unmarked nodes or nodes marked **U**, **O** or **I**

$\mathbf{VBD_P}$: These variables are associated only with nodes marked **B**

$\mathbf{I(P)}$: These distinct clauses label nodes marked **I**

$\mathbf{B_P}$: These edges are marked e_1

$\mathbf{LOUD_P}$: These edge labels associate with e_2 edges, e_3 edges and some $e_3 - e_4$ pairs

$\mathbf{LUS_P}$: These edge labels associate with e_3 edges not in $\mathbf{LOUD_P}$

$\mathbf{LBD_P}$: These edge labels associate only with e_1 edges

Figure 2: Illustration of terms used in Theorem 5