Research Summary

Douglas D. Edwards

December 4, 1996

Contents

1 Introduction and overview

The aim of my research is the design of efficient algorithms for the propositional satisfiability problem ("SAT") and its complement, the propositional unsatisfiability problem ("UNSAT"). Since SAT is NP-complete and UNSAT is coNP-complete, these problems are unlikely to have an exact polynomial-time algorithm. Thus, "efficient" must refer to expected average complexity, or even just to empirical observation of practically useful performance on a wide range of cases agreed to be important. Even when we lower our sights in this way, however, we are still left with an interesting, important, and unsolved problem. Like other theoretically intractable problems, SAT and UNSAT do not go away simply because we can produce evidence that they are intractable. SAT and UNSAT are at the boundary between artificial intelligence and theoretical computer science. But although I *use* theoretical intractability results, my perspective and ultimate goals are those of AI and efficient algorithm design.

Recently considerable progress on SAT has been made with methods like GSAT and WALKSAT that are based on local search and heuristic repair. These methods guess a random truth value assignment (TVA), take some measure of how close it comes to satisfying the given formula, modify it accordingly, and iterate. Local search methods, however, are *incomplete*: they do not enumerate the space of TVAs exhaustively, and thus cannot prove unsatisfiability. Because I aim at an efficient solution for UNSAT as well as for SAT, I focus instead on other methods, such as the Davis-Putnam procedure (DP), which are based on exhaustive case analysis and can prove unsatisfiability. Indeed, my focus is more on UNSAT than on SAT, since unsatisfiable formulas tend at present to be the hardest cases even for algorithms that can in principle solve them.

The Davis-Putnam procedure in its simplest version is a branching tree search based on case analysis: select a propositional variable, mark it true in one branch and false in the other, simplify, and recurse; the formula is unsatisfiable iff all cases thus generated are unsatisfiable. On each branch, a partial TVA is constructed by the successive assignments to variables. If a satisfying TVA is found at any point, the search is cut short and the satisfying TVA is returned.

Since DP is so central to my research, perhaps it is worthwhile to set it out in slightly more detail. We operate on a formula A in clausal form (conjunctive normal form, where the conjoined "clauses" are disjunctions of literals). Initially, A is the target formula to be tested for satisfiability. The procedure will report that A is unsatisfiable or return a TVA μ that satisfies A. (The TVA μ may be partial; if so, every total TVA that extends μ will satisfy A.)

1. If A is empty (representing a vacuously true empty conjunction), then A is satisfiable, and any partial TVA satisfies it; return an empty TVA.

2. If A itself is not empty, but one of its clauses is empty (representing a vacuously false empty disjunction), then report that A is unsatisfiable.

3. Otherwise, select a propositional variable p that occurs in A, and "split" on p: create two new formulas A^+ and A^- from A. In A^+, assign p the value true; in A^-, assign p the value false.

4. Simplify A^+ and A^-: remove any false literals from their clauses, and remove any clauses containing true literals.

5. Apply DP recursively to A^+ and A^- (in either order). (If the first subproblem tried yields a satisfying TVA, cut the search short and do not check the other subproblem.)

6. If either subproblem yields a satisfying TVA, extend it by assigning the corresponding value to p (true for A^+, false for A^-) and return the resulting new TVA.

7. If A^+ and A^- are both reported to be unsatisfiable, then report that A is unsatisfiable.

Another type of inference step available for SAT and UNSAT algorithms, in addition to the splits on variables that characterize DP, is *extension* or *definition*, which alters the formula being examined, either at the top of the DP search tree (thus affecting all inferences thereafter) or at some point on one of the branches. A definition step operates on two propositional variables a and b that are already present in the formula, and another new propositional variable c that is *not* already present in the formula. The formula is extended by conjoining to it the clausal form of $c \longleftrightarrow f(a, b)$, where f is a binary truth function or propositional connective. For example, where f is conjunction, we would add the clauses $\neg c \vee a$, $\neg c \vee b$, and $\neg a \vee \neg b \vee c$. Multiple definition steps may also be telescoped to define new variables using truth functions of more than two arguments. The new formula, with the definition conjoined, is a conservative extension of the original formula: the set of logical consequences not containing c does not change. However, the amenability of the formula to efficient solution by further DP steps may change radically. DP with definition steps is called *extended DP*.

The main degree of freedom in DP is the choice of variable to split on at each branch. If extensions are allowed to be freely intermixed with splits, more degrees of freedom are created, since at each inference step one may choose between splitting and definition, and the variables a and b and the truth function f must be selected at each definition step. One useful tool for choosing what inference step to perform is full-width one-step *lookahead*: every available split (and perhaps every available definition as well) is carried out, and the resulting pairs of formulas (for splits) or extended single formulas (for definitions) are evaluated according to some heuristic measure of difficulty of solution. (For pairs of formulas resulting from splits, this measure is likely to be defined by giving a heuristic measure of difficulty for the individual formulas and a rule for combining the results for the two formulas.) Then whatever inference step appears to yield the easiest formula or pair of formulas is committed to, and DP continues. The other trial splits or definitions used in lookahead are discarded.

Another available tool for search guidance in extended DP is *deferred splitting*. A trial split in lookahead may leave large portions of the formula identical on both sides of the split. If the best available split is committed to and carried out immediately, much work may be duplicated unnecessarily. Repeated duplication of effort in later splits may lead to unnecessary exponential increases in search time. So, instead of committing to the split and continuing extended DP directly, it would be possible to carve out a common subformula from both sides of the split in some manner and defer the best available split: save it on a stack or other data structure and carry it out *after* the subformula has already been solved. Deferred splitting might also be used to guide definition. One could carry out definition steps to define a single variable equivalent to the common subformula.

The Davis-Putnam procedure is a standard tool in SAT and UNSAT algorithms; many available algorithms are variants of DP. Extension, lookahead, and deferred splitting are, by contrast, highly unusual in this context. Researchers using DP have usually considered full-width lookahead to be excessively extravagant, preferring to use simpler and less computationally costly heuristics to select the variable to split on. Although the extension inference rule has been known for a long time, to the best of my knowledge it has only been used as a proof rule in theoretical contexts, never before as a tool for building algorithms to solve SAT and UNSAT. And deferred splitting

is new, although it is reminiscent of Ginsberg's dynamic backtracking (which, however, does not incorporate extension steps).

One can see why the high resource costs of extension, lookahead, and deferred splitting would cause algorithm designers to shy away from them. Extension is particularly dangerous; it allows arbitrary increases in the number of clauses and variables, thus making it difficult to find any theoretical upper bound at all on its time complexity, even an exponential bound. However, I believe that theoretical intractability results from the 1980s provide compelling evidence that many problems of interest will not be efficiently solvable by straight DP, regardless of the heuristics used for choosing the variable to split on. These results are particularly compelling because they are based on relatively natural examples and do *not* rely on the assumption that P is not equal to NP: they are, instead, outright exponential time complexity results specifically for DP and related algorithms.

In my judgment, this field of research is not lacking in powerful heuristics to guide search. Instead, the problem is that the space being searched often does not contain any good solutions: DP can only generate exponential-size trees. Since the other main approach to SAT (local search through the space of TVAs with heuristic repair) is not applicable to UNSAT at all, I believe we need to try to tame extended DP in order to make progress, and that effort is the focus of my research. It is better to search a vast space that probably contains solutions, rather than a smaller but still intractably large space that probably does not.

In the next sections, I will describe the history of the SAT and UNSAT problems and the currently popular approaches, leading up to a discussion of the intractability results and a critique of the existing approaches. Then I will report on the status of my experiments with lookahead (for straight DP) and my efforts to design efficient algorithms using extended DP with lookahead and deferred splitting.

2 History of the propositional satisfiability problem

2.1 Frege systems and resolution

Since propositional validity is unsatisfiability of the negation, any proof system for propositional logic is applicable to the UNSAT problem. Propositional inference rules date back to antiquity, and interest in them was revived by Boole in the mid-19th century, but the first correct and complete axiomatic system for propositional logic was given in *Begriffsschrift* (Frege, 1879), as part of the first correct and complete axiomatic system for *first-order* logic. Correct and complete axiomatic systems for propositional logic based on sets of connectives adequate to define all Boolean functions are called *Frege systems*. Remarkably, although Frege systems are defined without reference to proof length or any other measure of efficiency, any Frege system can polynomially simulate any other with respect to length of proofs (Cook and Reckhow, 1979). Frege systems enhanced with the extension rule are called *extended Frege systems* or *EF-systems* and are maximally powerful among known propositional proof methods with respect to length of proofs.

SAT and UNSAT have been seminal problems for both artificial intelligence and theoretical computer science. The first AI program that ever actually ran on a computer, the Logic Theory Machine (Newell et al, 1957), addressed UNSAT using a Frege system. However, Frege systems have high branching factors and can generate tautologies from their own logical axioms in free-association style, without reference to the problem at hand. For these reasons, more efficient and

focused methods were soon sought. These methods had no logical axioms and were based on the *resolution* inference rule: from $a \vee b$ and $\neg b \vee c$ infer the *resolvent* $a \vee c$, which does not contain the variable b. The expression b is said to be *resolved on*. (Resolution is more natural than it looks: it is equivalent to either of the more natural-looking inference rules *hypothetical syllogism* (from $a \longrightarrow b$ and $b \longrightarrow c$ infer $a \longrightarrow c$) and *constructive dilemma* (from $b \longrightarrow a$ and $\neg b \longrightarrow c$ infer $a \vee c$). Any of these three rules can be derived from any other by substitution for variables, commutativity of disjunction, and use of the definition of $p \longrightarrow q$ as $\neg p \vee q$.) Resolution cannot derive anything without premises. Thus, it is not complete, and does not in itself constitute a Frege system. However, it is *refutation complete* for formulas in clausal form: that is, given the clauses of any unsatisfiable formula in clausal form as premises, resolution can derive the patently contradictory empty clause.

As is well known, Robinson (1965) combined resolution with unification to yield an efficient proof procedure for first-order logic. However, he did not invent resolution, nor was he the first to apply it to automated deduction. Resolution appeared first in a theoretical context (Baker, 1937), and the earliest form of the DP algorithm (Davis and Putnam, 1960) was a variant of resolution. This "DP60" algorithm did not branch like the version of DP set out above, which is the one used today. Instead, it used resolution as an inference rule to accumulate more clauses in a linear sequence while progressively eliminating each variable along with the clauses that contained it. Instead of branching, the split variable was eliminated by replacing the set of clauses containing it with the set of all their possible resolvents. This exhaustive resolution procedure can easily generate huge numbers of clauses, and for this reason was quickly abandoned in favor of the branching variant set out above, called DPLL (Davis et al, 1962). DPLL can also be regarded as a thinly disguised resolution refutation procedure: each split can be viewed as a deferred resolution step. Recursively, resolution refutations (derivations of the empty clause) are generated for A^+ and A^-. Restoring the missing variable p converts the refutation of A^+ into a derivation of p from A and the refutation of A^- into a derivation of $\neg p$ from A. A final resolution step then yields the empty clause and completes a resolution refutation of A (Goldberg, 1979). For this reason, a DPLL proof of unsatisfiability of A must contain at least as many splits as there are inference steps in the shortest possible resolution refutation of A. All of the later intractability results are given for resolution, but it is important to note that they apply immediately to DPLL also.

2.2 Unit propagation and purity checking

There are two optimizations that were used with DP60 and DPLL from the beginning and are still nearly always used, because they often allow simplification without branching. Before applying DP recursively to A^+ or A^-, these simplifications are carried out iteratively until neither applies.

The first simplification is *unit resolution*, which is often called *unit propagation* in this context. If the formula contains a *unit clause* (that is, a clause consisting of a single literal p or $\neg p$), the clause can obviously only be made true by one of the two possible assignments of truth value to the variable p, so that assignment is made without branching and the resulting formula is simplified. This procedure is equivalent to resolving the unit clause with all other clauses containing p. Note that unit propagation by itself (actually even just propagation of only *positive* unit clauses) is a decision procedure for HORNSAT, that is, satisfiability for formulas consisting of *Horn* clauses containing at most one positive literal. Unit propagation decreases the number of clauses in a formula, so it must terminate. The set of Horn clauses is closed under unit propagation, and a non-

empty, non-unit Horn clause must contain at least one negative literal. Thus, if unit propagation terminates without yielding the contradictory empty clause, then the formula can be satisfied by marking all remaining variables false.

The other simplification is *purity checking* or *pure variable elimination*. A variable is *pure* in a formula if the literals containing it are either all positive or all negative. A formula containing a pure variable p can be simplified by assigning a truth value to p so that all literals containing it come out true, and then eliminating all clauses in which p occurred.

2.3 Probability models for random formulas

SAT was the first problem to be proved NP-complete (Cook, 1971). It follows that UNSAT is coNP-complete, so that polynomial-size proofs of unsatisfiability cannot even exist for all formulas unless NP = coNP, an eventuality generally considered hardly more likely than P = NP. But these results were not enough to dampen initial optimism about average-case tractability of SAT and UNSAT. Goldberg (1979) used the "constant probability" or "fixed density" model for random formulas, according to which each clause is created by giving each literal a fixed probability of being included, independently of the inclusion of other literals. With this model, Goldberg showed that DPLL has $O(cv^2)$ average case time complexity, where v is the number of variables and c is the number of clauses. But the constant probability model was discredited by Franco and Paull, who showed that Goldberg's distributions produced formulas that were so easily satisfied that simply guessing random TVAs will find a satisfying TVA within a constant number of guesses with probability asymptotically approaching 1 (Franco and Paull, 1983; Franco, 1986).

Franco and Paull suggested the k-SAT or "fixed clause length" model might produce harder and more realistic random formulas. In this model, all clauses have exactly k literals for some fixed k, and are produced by sampling k times from a uniform distribution over the $2v$ available literals. 2SAT is solvable in linear time, but for $k > 2$, k-SAT is NP-complete in theory and many of its random instances under this distribution are quite hard.

3 Current approaches to satisfiability

3.1 The phase transition phenomenon in random 3SAT

After the constant probability debacle, SAT and UNSAT research has focused strongly on finding genuinely hard examples. Random 3SAT has been a popular choice. An excellent recent survey is (Cook and Mitchell, 1996).

One particularly intriguing phenomenon is the empirically observed "phase transition" in random k-SAT, which depends on the clause-to-variable ratio c/v. Below a certain fairly sharp threshold region, which depends strongly on the clause length k but only very weakly on the number of clauses c (empirically about $c/v = 4.24$ for 3SAT with very large c), formulas are almost always satisfiable, with a high density of satisfying TVAs, and are very easy for DPLL. Within the threshold region, formulas are about equally likely to be satisfiable or unsatisfiable, and in either case tend to be very difficult for DPLL. Above the threshold region, formulas are almost always unsatisfiable, and once again become *relatively* easy for DPLL, at least compared to those in the threshold region, although they never get as easy as those well below the threshold region. This pattern tends to hold empirically for other algorithms as well as for DPLL. It even holds for the

incomplete local search methods when only satisfiable formulas are considered. Even when two algorithms differ greatly in their overall performance on 3SAT (local search tends to work much better than DPLL for satisfiable formulas in the threshold region), relative difficulty of problems *for each algorithm considered separately* tends to show the same pattern depending on c/v, with the same narrow threshold region.

A great deal of current research is based on the hypotheses that:

- Clause-to-variable ratio c/v is an objective, algorithm-independent measure of absolute difficulty for random k-SAT instances in some important sense.
- The threshold region for c/v is amenable to study by the same techniques used to analyze phase transitions in physics.

Recently the journal *Artificial Intelligence* came out with an entire "Special Volume on Frontiers in Problem Solving: Phase Transitions and Complexity" (Hogg et al, 1996), devoted to this approach.

3.2 Local search and heuristic repair

The great difficulty of random 3SAT in the threshold region for DPLL has not stopped UNSAT research based on DPLL, but it has given increased prominence to incomplete algorithms based on local search and heuristic repair, which only work on satisfiable instances. This latter subfield was inaugurated by GSAT (Selman et al, 1992). GSAT starts with a random TVA. It then does a form of lookahead by "flipping" each variable in the formula in turn (reversing its polarity) and then counting the number of clauses in the formula that are not satisfied by the resulting TVA. It uses this count as a heuristic measure of distance to the goal, and carries out a local hill-climbing search, committing to the flip that resulted in the fewest unsatisfied clauses and continuing the search from the resulting TVA. GSAT differs from standard hill-climbing in that it does not give up or restart from a random TVA whenever it reaches a local minimum; instead, it picks the best flip it can find and continues at all costs, even if the best available flip sends it to a TVA with a number of unsatisfied clauses equal to, or even greater than, that of the current TVA. Experiments show that GSAT's willingness to glide across plateaus in the search space in this way is crucial to its success. However, GSAT does set a maximum number of allowed flips per local search attempt, and will restart at a new random TVA and try again whenever it has reached the allotted maximum. It also has a parameter setting a maximum for the total number of local search attempts, and when it reaches that maximum it simply gives up.

GSAT can run *much* faster than DPLL on many satisfiable formulas, and for a while it was thought that it might dominate DPLL on all satisfiable formulas. However, Konolige (1994) has found a class of satisfiable formulas based on an abstraction from crossword puzzles that is difficult for GSAT but easy for DPLL. Still, it is fair to say that, on the whole, local search methods are superior in the current state of the art for most classes of satisfiable formulas, especially since the target formula is often strongly suspected of being satisfiable, or even known to be satisfiable, and the real task is to find a specific satisfying TVA.

Currently the most effective local search and heuristic repair algorithm is WALKSAT (Selman et al, 1994), which does not do a full-width lookahead as GSAT does, but instead selects a random clause *not satisfied by the current TVA*. It then randomly either:

- Picks a variable from the chosen clause at random and flips it, or:

- Does a limited lookahead among only the variables occurring in the chosen clause, and flips the one that leads to the fewest unsatisfied clauses under the resulting TVA.

WALKSAT can solve, in about one hour each, 50% of random 3SAT instances in the hardest part of the threshold region ($c/v = 4.24$) for $v = 2000$ variables. This probably means that it is scoring nearly 100% on satisfiable instances only, since the other 50% of the instances are probably unsatisfiable. No currently available complete method is efficient enough to prove their unsatisfiability within a reasonable amount of time; the best versions of DPLL can handle no more than about 400 variables for formulas in the hardest part of the threshold region within any reasonable time limit.

Although this performance by WALKSAT is impressive, there is one challenge problem, involving only satisfiable formulas and not even known to be NP-hard, that neither complete nor incomplete methods can currently handle: factoring products of large primes. This task is particularly illuminating as to the limits of the state of the art, because here SAT and UNSAT solvers (working on multiplier circuits encoded as propositional formulas) must compete with specialized algorithms based on number theory. Cook and Mitchell (1996) propose factoring as a challenge problem, and state that current number-theoretic methods can handle about 200 bits (about 60 decimal digits) for each prime, and current SAT and UNSAT solvers are far below this level of proficiency. Even though incomplete methods currently have the lead over complete methods on SAT, the race between the two kinds of methods is not over, and no current method does well enough for us to pronounce SAT (much less UNSAT!) efficiently solved for practical purposes.

Incidentally, although the current local search and heuristic repair algorithms that search the space of TVAs cannot, of course, deal with unsatisfiable instances, Selman (1995) has suggested that this difficulty could be overcome by searching in a space of proofs instead. I am not pursuing this intriguing suggestion at present, but it might be a promising path for other researchers.

3.3 Split variable selection heuristics for DPLL

Current heuristics for splitting variable selection in DPLL, such as that used in the state-of-the-art TABLEAU implementation (Crawford and Auton, 1996), tend to be based on combinations and variants of two heuristics:

- The small-clause heuristic, used originally in (Davis and Putnam, 1960) and developed further in (Chao and Franco, 1986, 1990). Pretolani (1993) invented the mnemonic *Moms* for this heuristic: split on the variable with the **M**aximum number of **o**ccurrences in **m**inimum **s**ize clauses. In practice, this often means splitting on the variable that occurs most often in binary clauses.

- The Jeroslow-Wang heuristic (Jeroslow and Wang, 1990), which is quite similar in spirit to the full-width one-step lookahead approach, but in effect attempts to predict the result of a lookahead without actually carrying it out. For each variable and each truth value that could be assigned to it, the heuristic estimates the probability that the resulting formula would be satisfiable, according to syntactic measures based on clause length. Like other forms of lookahead, Jeroslow-Wang needs a combination rule to transform a pair of estimates for the two formulas resulting from a split into a single estimate for the split itself. The rule used is satisfiability maximax: the score given to a split is the higher of the two satisfiability

estimates for its resulting formulas. TABLEAU attempts to improve on Jeroslow-Wang by also estimating the number of unit propagations that will result from a split, but still stops short of doing full lookahead with unit propagation and purity checks on each resulting formula (Crawford and Auton claim that it is not cost-effective).

Besides heuristics for split variable selection, another recent contribution to the field has been *dynamic backtracking* (Ginsberg, 1993; Ginsberg and McAllester, 1994). Like the old AI technique of dependency-directed backtracking (Stallman and Sussman, 1977), dynamic backtracking allows earlier splits to be revoked and the search tree rearranged on the fly, so as to escape from dead ends created by unfortunate choices high in the tree without undoing all the intervening work (which may not actually depend on the split being revoked). But dependency-directed backtracking risked slowing down the search and exhausting memory by accumulating dependency information indiscriminately. Dynamic backtracking, by contrast, uses sophisticated forgetting mechanisms to try to ensure that only a polynomial-size amount of information is ever retained at any one time. Neverthless, unfortunate interactions with search heuristics can lead to exponential increases in the size of the search space under dynamic backtracking (Baker, 1994).

3.4 Lookahead in DPLL

So what are we to make of all this? At one point, I felt that in order to advance the state of the art in DPLL, we merely needed full-width one-step lookahead with unit propagation and pure variable elimination on all the resulting formulas, along with more effective search heuristics to make it worth the trouble (contrary to Crawford and Auton's previous experience). One great advantage of lookahead is that (modulo the need for a combination rule) it allows any measure of the difficulty of formulas to be converted directly into a heuristic to guide search. I proposed using c/v itself as a heuristic, with a weighting scheme similar to that used in Jeroslow-Wang to deal with clauses of different sizes. Preliminary experiments with medium-sized random 3SAT instances in the hard region indicated that lookahead tended almost to pay for itself regardless of the effectiveness of the heuristic, since at least 90% of branch points where the lookahead process was initiated turned out to be *singular choice points* where one of the trial splits done for lookahead yielded a contradiction on one of the branches. Whenever this happens, the lookahead can be cut short without doing any pending trial splits, and the split where one branch has yielded a contradiction can be committed to immediately. No actual branching occurs, however, since the contradictory branch has already been closed off, and only the other branch need be followed. The effect is like that of a unit propagation or a pure variable elimination.

I also felt there was considerable room for more sophisticated heuristics. The seldom-noticed paper (Jackson, 1992) shows that for random 3SAT instances, the "cubic sparsity" measure c/v^3 is a better predictor of the time complexity of solving an instance than straight c/v, for an algorithm similar to DPLL. For random 3SAT, in fact, cubic sparsity works so well that, given a known, fixed value for the cubic sparsity of an instance, its difficulty becomes *independent* of its size (the number of clauses c)! An appropriate modification of cubic sparsity to allow for clauses of varying sizes might be a powerful enough heuristic to justify the $O(cv)$ worst-case cost of full-width lookahead with unit propagation and purity checking, even disregarding the empirical evidence that this full cost often need not be paid in practice. (The v factor comes from checking every variable, and the c factor comes from the worst-case cost of unit propagation and purity checking, which may affect large numbers of clauses.)

I still believe that the research program just described would hold promise, if simply improving the state of the art in DPLL were an appropriate goal. However, fuller appreciation of the theoretical intractability results for resolution and DPLL has led me to believe that far more radical changes are needed if we are to make major progress in UNSAT and that the current focus on the random 3SAT phase transition is distracting attention from far more fundamental difficulties. In the next section, I turn to the history of the intractability results.

4 Intractability results for resolution and DPLL

4.1 Expander graphs and the pigeonhole principle

The historical sequence of the intractability results starts with (Tseitin, 1968). In this paper the notion of a *regular* resolution refutation is defined. This is a resolution refutation in which no clause is resolved on the same variable twice. Tseitin demonstrated (although with notorious lack of detail) a superpolynomial worst-case lower bound for regular resolution using certain sets of clauses generated from labeled graphs.

Although Tseitin's complexity result, and the later improved results derived using the same model, depend on the specific graphs used, the labeling system is always the same. Each edge of the graph is labeled with a literal and each vertex of the graph is labeled with a *charge* of 0 (representing falsity) or 1 (representing truth). The set of clauses corresponding to a vertex labeled with charge x and having incident edges labeled with $q_1, \ldots, q_k$ is the clausal form of the equation $q_1 \oplus \ldots \oplus q_k = x$, where the plus sign represents exclusive or (in the logical interpretation) or addition modulo 2 (in the arithmetical interpretation). These clauses are all those having exactly one occurrence of each of the literals $q_1, \ldots, q_k$, with the polarity of some of the literals reversed, where the number of literals with reversed polarity has parity opposite to x. The set of clauses corresponding to the entire graph is simply the union of the clauses corresponding to the individual vertices.

Note that Tseitin's result already shows worst-case intractability of DPLL, since resolution refutations corresponding to DPLL proofs of unsatisfiability are regular. In the same paper, Tseitin introduced extended resolution (resolution plus the extension rule) and noted that his intractability result does not apply to it. In fact, none of the intractability results apply to extended resolution. The extended resolution proof system and extended Frege systems can polynomially simulate each other, so extended resolution belongs the strongest known class of proof systems.

Galil (1977) filled in many of the gaps in Tseitin's proof and improved the lower bound by using *expander graphs*. These are bipartite graphs with limited-degree vertices and the same number n of vertices on each side, with an *expansion factor* d having the following property: any set of $k = n/2$ or less vertices from one side of the graph is connected to at least a total of $(1 + d)k$ vertices on the other side. Galil's proof was simplified further in (Ben-Ari, 1980).

Haken (1985), using some ingenious counting techniques and a different class of examples, showed a superpolynomial lower bound for general resolution, including non-regular resolution (although not extended resolution). Haken's example was the propositional *pigeonhole principle* of arbitrary size n, or rather its (unsatisfiable) negation, which states that $n + 1$ pigeons are in n holes and no two are in the same hole. (Or, equivalently, that n pigeons occupy all of $n + 1$ holes without leaving any empty holes, even though a pigeon can fit in only one hole at a time; this interpretation gives rise to the same set of clauses.) Taking $P(i, j)$ to mean that pigeon i is in hole

j (or, in the other interpretation, that hole i contains pigeon j), the negated pigeonhole principle $\neg\mathrm{PHP}(n)$ contains the $O(n^2)$ clauses $P(i,1) \vee \ldots \vee P(i,n)$ for all i from 1 to $n+1$, and the $O(n^3)$ clauses $\neg P(i,j) \vee \neg P(k,j)$ for all j from 1 to n, all i from 1 to $n+1$, and all $k > i$ up to $n+1$. (The variables i, j, and k are, of course, metalinguistic indices only; no object-level quantificational logic is used, and the proof systems in question here treat the $P(i,j)$'s as unrelated, atomic propositional variables.)

Urquhart (1987) improved Haken's worst-case lower bound on general resolution to a true exponential c^n for a constant $c > 1$, which is the best possible bound up to the choice of the constant c, by applying Haken's counting techniques to expander graphs similar to Galil's. The bound was improved because Urquhart needed only $O(n)$ of these graph-based clauses, rather than the $O(n^3)$ clauses of $\neg\mathrm{PHP}(n)$. Incidentally, (Urquhart, 1987) is by far the most clearly written and easiest to understand of these papers dealing with the intractability of resolution and DPLL, and Urquhart explains much of the material of the earlier papers. Anyone previously unfamiliar with this topic who wishes to explore it should start with Urquhart's paper.

4.2 Sparsity properties in random hypergraphs

The last and strongest of the intractability results is from (Chvátal and Szemerédi, 1988). They generalize the properties of Urquhart's expander graphs to certain "sparsity" properties (unrelated, incidentally, to Jackson's "cubic sparsity") which classes of random hypergraphs corresponding to random k-SAT instances are shown to possess. (Any set of clauses can be viewed as a hypergraph by treating variables as vertices and clauses as edges, and disregarding the polarity of literals.) Using these properties, they demonstrate that random k-SAT formulas with v variables and c clauses, where $c/v > 0.7 \cdot 2^k$ and $k > 2$, are intractable for general resolution in a strong sense: there exists a real constant $b > 1$ such that, with probability tending to 1 as v tends to infinity, a random instance is unsatisfiable but its shortest resolution refutation contains at least b^v clauses.

For random 3SAT, the condition on clause-to-variable ratio works out to $c/v > 5.6$, which, ironically, places the lower limit for Chvátal and Szemerédi's result well into the supposedly "easy" overconstrained region, far beyond the "hard" phase transition threshold region around $c/v = 4.24$. While there is no reason to doubt the empirical findings that formulas in the threshold region are even harder, Chvátal and Szemerédi's result does indicate quite strongly that the "easy-hard-easy" pattern suggested by empirical research is highly misleading; "easy-hardest-hard" is more like it. To the extent that random 3SAT is to be taken seriously at all as a model for the real-world problems to which a practically efficient UNSAT algorithm would ultimately be applied, it would seem that the right conclusion to draw is not that the phase transition should be investigated intensively to find heuristics to speed up DPLL, but rather that something stronger than DPLL will be needed and the investigation of straight DPLL should be abandoned in favor of a search for that "something stronger." That is the conclusion I draw and the direction my research is taking.

4.3 Is random 3SAT a realistic model?

Although much of the community studying SAT and UNSAT algorithms does seem to believe that random 3SAT is a good model for practical problems, that viewpoint is far from obvious and there are prominent dissenting voices. Ginsberg and McAllester (1994) did not even test their partial-order dynamic backtracking algorithm against random 3SAT, citing evidence from (Crawford and Baker, 1994) that real-world problems (scheduling problems, in particular) behave

differently from random 3SAT problems. Crawford and Baker found that scheduling problems tend to be satisfiable and highly underconstrained (though quite large), yet neither WALKSAT nor the TABLEAU implementation of DPLL did well on them. Scheduling problems tend to contain long chains of variables whose values depend very strongly on a few "control" variables, so that the number of actual degrees of freedom in a formula is very much smaller than the number of variables. This is the same property that the hard examples for GSAT from (Konolige, 1994) had. Because WALKSAT treats each variable independently in measuring how close the current TVA comes to satisfying the given formula, it flounders on such problems. TABLEAU, like other forms of DPLL, uses unit propagation and so is able to detect such chains of dependent variables. But TABLEAU ran into a different problem: it could easily fall into what I call "black holes" (their term is "deserts" devoid of solutions). Even though scheduling problems have many satisfying TVAs, a few badly chosen splits at the top of the search tree could eliminate all the solutions and land TABLEAU in a difficult unsatisfiable subproblem, where it would thrash indefinitely. The algorithm that worked best was "iterative sampling," which randomly selects truth values for variables one at a time (rather than guessing a whole TVA at once as in the initialization phase of WALKSAT), and performs unit propagation like DPLL each time a variable is set, but does not backtrack: instead, it simply gives up and starts again whenever a contradiction is discovered.

Iterative sampling is certainly a reasonable way to finesse the "black hole" problem when dealing with highly underconstrained practical problems where the goal is to find any one of the many available satisfying TVAs. However, it would be quite a leap to conclude that unsatisfiable random 3SAT problems are irrelevant to UNSAT research. I would like to know what those "black hole" unsatisfiable subproblems looked like. Did *they* look like critically constrained or overconstrained random 3SAT problems?

Or, perhaps, did they look like pigeonhole or expander graph problems? One reason I am convinced that the intractability results require us to look to proof systems more powerful than plain resolution is that the pigeonhole examples, in particular, look as though they could easily come up in practice. Substitute "required tasks" for "pigeons," and "remaining available resources" (after some unfortunate assumptions about allocation of other resources) for "pigeonholes," and we get a highly plausible "black hole" scheduling subproblem. The underlying difficulty is that there are too many pigeons and not enough pigeonholes, regardless of how they are allocated, but resolution cannot do much better than to try and reject every possible assignment of pigeons to holes individually. Bart Selman (personal communication) believes that hidden pigeonhole substructure is a common source of intractability in realistic UNSAT problems.

Extended resolution or extended DP, by contrast, can define new variables that express concepts like "At least k of the first r pigeons are in the first s holes" (according to some arbitrary ordering imposed on the pigeons and the holes) and can efficiently mimic our intuitive understanding of why the principle must be true. Perhaps more surprisingly, Frege systems can also prove the pigeonhole principle in polynomial size, even without the extension rule (Buss, 1987). (These results refer to the *existence* of short proofs, not the ease of finding them.) Whether or not random 3SAT is accepted as an accurate model of realistic UNSAT problems, we cannot rest content with a model of propositional proof that cannot generalize to the extent of being able to show efficiently that requirements exceed resources in the pigeonhole examples.

Even the expander graph problems are more natural than they look. It is true that the authors of the intractability papers using expander graph examples must resort to highly arcane graph-theoretic assumptions and ingenious counting techniques to show intractability. However, it is

only the weakness of resolution inference that makes the structure of the graph relevant. The underlying problem is quite simple: resolution cannot handle reasoning with XOR (or, equivalently, with biconditionals) efficiently.

Informally, we know that XOR is associative and commutative, and that $p \oplus p$ is always false. Using these facts, we can see that a graph-based clause set is unsatisfiable if the sum of the charges on the vertices modulo 2 is 1, *regardless* of the structure of the graph, as follows. Every literal occurs in an even number of the equations $q_1 \oplus \ldots \oplus q_k = x$, for the various graph vertices, because each of the one or more edges labeled with that literal is attached to precisely two vertices. Thus, the sum mod 2 of the left-hand sides of the equations corresponding to the vertices (the truth-value of their iterated XOR) must be 0. On the other hand, by hypothesis the right-hand sides (the charges on the vertices) add up to 1 mod 2; contradiction. Again, extended resolution can in effect duplicate this reasoning and cut through all the complexities that these graph-based formulas generate for plain resolution. And again, any Frege system can do the same, even without the extension rule.

Expander-graph type problems may not show up as subproblems for DPLL in solving scheduling problems, but the multiplier-circuit clauses derived from factoring problems use lots of XORs in their adder subcircuits. Any proof method that has problems with expander-graph clauses is likely to fail miserably on multiplier circuits. And a retreat from DPLL to iterative sampling is unlikely to help either, since factoring problems of the kind one would be interested in solving generally have only one solution.

5 Capitalizing on extended DP

Having explained why I think the extension rule must be used and tamed in spite of its potential for combinatorial explosion, I now set out the research directions available to be pursued. I present three approaches:

1. Using hypergraph sparsity properties as heuristic measures.
2. 2SAT subproblems and splits on complex formulas.
3. Subproblem creation via deferred splitting.

All three approaches are described in order to show the robustness of the research program. The goal of taming extended resolution gives rise to a number of interesting approaches, and the failure of any one approach will not kill off the research program, since there are a number of other approaches in reserve. The presentation of a variety of approaches should not, however, be taken to indicate that I regard all of these approaches as equally promising or that I intend to pursue all of them in parallel without setting priorities. The third approach, subproblem creation via deferred splitting, appears by far the most promising at present. It is the major focus of my current research, and is the approach most likely to yield a algorithm that can efficiently find short proofs for the pigeonhole and graph-based examples and for hard random 3SAT problems.

5.1 Using hypergraph sparsity properties as heuristic measures

The most straightforward way to incorporate definition steps into DPLL is to consider them as alternatives to splitting and include trial definitions as well as trial splits in a lookahead step. To

keep the definition lookahead down to $O(v^2)$, only single definition steps (using pairs of existing variables on the right-hand side of the split) would be considered. There are sixteen binary truth functions. Of these, six are trivial (constant, or equivalent to a function of one input variable). Of the ten remaining nontrivial binary truth functions, we can select five that can be expressed as binary clauses or pairs of binary clauses, and the other five are just their negations and can be omitted. So for each pair of input variables a and b, we need to check the definition of a new variable c as one of the five functions $a \vee b$, $\neg a \vee b$, $a \vee \neg b$, $\neg a \vee \neg b$, or $a \longleftrightarrow b$ (that is, $(\neg a \vee b) \wedge (a \vee \neg b)$).

One could try using with extended DPLL any of the lookahead heuristics that can be used with ordinary DPLL. However, the use of definition steps which extend the formula conservatively, changing its syntactic structure without altering its set of satisfying TVAs, does point up some weaknesses in the available heuristics, which tend to be geared to random formulas without the kind of redundancy that definitions introduce. For example, under certain plausible extensions of Jackson's cubic sparsity heuristic to clauses of varying size, any definition step carried out on a random 3SAT problem will always decrease its cubic sparsity. If the heuristic were taken at face value, that would mean that definition always makes such problems harder.

On the other hand, we could look to the research on intractability of resolution for clues to better heuristics. In particular, we might try to use the hypergraph sparsity properties of Chvátal and Szemerédi (unrelated to Jackson's cubic sparsity) as measures of the difficulty of a formula.

As it happens, it appears to be difficult to use the hypergraph sparsity properties directly as heuristics. Consider the first and simplest of them, "(x, y)-sparsity." A hypergraph with v vertices is (x, y)-sparse, where x and y need not be integers, if every set of s vertices, where s does not exceed xv, is connected by at most ys hyperedges. This property is itself coNP-complete: CLIQUE reduces easily to a special case of the complementary property (where, among other things, the hypergraphs are restricted to be ordinary undirected graphs). Also, as it stands, (x, y)-sparsity is a Boolean property, not a continuous measure.

Another sparsity property is "property $P(a)$." This property is concerned with the size of the *boundary* of a set of hyperedges, defined as the set of vertices to which precisely one hyperedge in the set is incident. A hypergraph with v vertices has property $P(a)$ if every set of hyperedges of size m not exceeding av has a boundary of size at least $m/2$. Again, this property is Boolean, and it looks difficult to check. Worse yet, the hypergraph corresponding to any formula containing even one definition of the form $r \longleftrightarrow f(p, q)$ for a nontrivial binary Boolean function f will lack this property for any but the smallest values of a. Each of the variables (corresponding to vertices) p, q, and r will occur in at least two of the clauses (corresponding to hyperedges) of the definition, so that these clauses will form a counterexample to the property.

Nevertheless, modifications of these properties could be used to guide search. They could be converted into continuous measures by measuring the *proportion* of small sparsely connected vertex sets, or small sets of hyperedges with large boundaries, instead of requiring *all* sufficiently small sets of vertices or hyperedges to have the relevant properties. These continuous measures would, however, probably still be intractable to compute. But it might be easier to measure the *difference* in these sparsity measures before and after a given trial definition or trial split than to compute the absolute sparsity measures themselves, especially since a definition or split generally only affects a small subset of variables or clauses.

5.2 2SAT subproblems and splits on complex formulas

Defining a new variable as a nontrivial binary Boolean function of two existing variables, and then immediately splitting on the new variable, has the effect of splitting directly on the complex formula (consisting of one or two clauses) defining the Boolean function. This limited use of definition is particularly attractive since the definition and the split can be telescoped into a single atomic inference step, so that some of the power of definition is retained but the actual introduction of new variables is avoided altogether. Also, a split on a complex formula can do more than a split on a single existing variable. Splitting on $\neg a \vee b$, for example, adds this binary clause to the current formula on the true branch; on the false branch, it simultaneously marks a true and b false, along with any ensuing unit propagation and pure variable elimination. Splitting on the biconditional $a \longleftrightarrow b$ has a slightly different effect. It allows a reduction in the number of variables (on both branches) by equating a with b on the true branch and equating a with $\neg b$ on the false branch. Such splits on disjunctions or biconditionals could enhance the power of splitting and lookahead without the explicit use of definition.

Splitting on complex formulas representing nontrivial binary Boolean functions is particularly appealing when used together with the known but little-used technique of adjoining a specialized algorithm for efficient solution of 2SAT subproblems to DPLL. Larrabee has experimented with treating the binary clauses in a formula as a separate 2SAT subproblem to be solved using a digraph algorithm (Aspvall et al, 1979), successively enumerating the partial TVAs satisfying this 2SAT subproblem, and checking them against the non-binary clauses (Larrabee, 1992; Larrabee and Tsuji, 1992). Richard Karp (according to Selman, personal communication) has proposed using an efficient 2SAT algorithm not to enumerate satisfying partial TVAs but to check for:

- Unsatisfiability of the 2SAT subproblem, which of course implies unsatisfiability of the entire current formula and terminates the current branch;
- Unit clauses implied by the 2SAT subproblem, which can initiate simplification via unit propagation;
- Equivalences between variables (or, more generally, literals) implied by the 2SAT subproblem, which can allow reduction in the number of variables on the current branch.

Splitting on binary clauses and biconditionals would further enhance the power of Karp's suggested uses of 2SAT subproblems. We would not have to wait passively for binary clauses to be given in the top-level formula or created by splits and unit propagations. Instead, we could aggressively introduce them whenever lookahead (or any other type of clue) indicated that they would be helpful.

Moreover, graph-theoretic properties of the digraph representation of the existing 2SAT subproblem could suggest helpful binary clauses to try. For example, variable equivalences (including the impossible equivalence between p and $\neg p$, which indicates unsatisfiability) are detected via cycles in the digraph. So a binary clause that would complete a large cycle would be a good choice for a split. Even in the absence of so direct an indicator, one could aim generally to increase the connectivity of the 2SAT subproblem, which is often not high. For example, in the pigeonhole problem, where a large number of binary clauses are given at the outset, each of the pigeonholes is represented in the initial 2SAT digraph as a weakly connected component, completely disconnected from the other pigeonholes, and each weakly connected component is a bipartite digraph with all the arcs running from $P(i,j)$ to $\neg P(k,j)$ for all k not equal to i.

The use of a specialized 2SAT subproblem solver along with splits on binary clauses looks more elegant than other parts of my research program (and combines well with them). Unfortunately, by itself this elegant idea does not yet seem to yield much headway against the pigeonhole problem. The fundamental difficulty is the same as with the use of definition in lookahead and the use of hypergraph sparsity properties to guide search. These approaches all work bottom-up, starting from individual variables given in the problem statement. For the pigeonhole problem, the natural recursive decomposition of the problem is top-down: we reduce $\neg\mathrm{PHP}(n)$ to $\neg\mathrm{PHP}(n-1)$. Although it may yet happen that the bottom-up approaches will somehow find the right starting point for this recursion, namely $\neg\mathrm{PHP}(1)$, I do not at present see how. In the next subsection I explain an approach which is somewhat less tidy and elegant than the use of 2SAT and splits on binary clauses, but which looks more promising as a path of attack on the pigeonhole examples.

5.3 Subproblem creation via deferred splitting

Although our ultimate goal is to have our programs find proofs for us, in order to get started we need examples, and that requires us to find proofs for our programs. Haken (1985) gives a polynomial-size proof of the pigeonhole principle using extended resolution, but I approached the problem from scratch and found my own proof using extended DPLL. (A proof was given much earlier in (Cook, 1976), but I have not yet been able to obtain that paper.) Although my proof is more complex than Haken's, in some ways it is an improvement. It requires only $O(n^3)$ definitions, in contrast to Haken's which requires $O(n^4)$ definitions. Also, I believe it is closer to the structure of an intuitive proof of the pigeonhole principle using unrestricted expressive resources (first-order logic with equality, etc.) My goal is to develop a search guidance method for extended DPLL that will find this proof automatically and in a way that allows generalization to other problems.

The proof (which I only sketch very broadly) uses new, defined variables with the following intuitive meanings and formal definitions, which refer to an arbitrary numerical order (starting at 1) imposed on the pigeons and the holes. Again, no object-level quantifiers are used and the indices are strictly for metalinguistic convenience in describing the proof. We may note, however, that one of the main strengths of the extension rule is that it provides an ersatz for parts of first-order predicate logic used in metalogical reasoning about proofs.

$R(r,s)$
: "Pigeon r is in one of the first s holes." This is just $P(r,1) \vee \ldots \vee P(r,s)$, but it can also be defined straightforwardly by recursion on s: $R(r,s) \longleftrightarrow R(r,s-1) \vee P(r,s)$. Note that the base case $\neg R(r,0)$, which is a unit clause, is a perfectly legitimate definition; it defines $R(r,0)$ as the constant identically false Boolean function. This would be optimized away by unit propagation anyway.

$Q(r,s)$
: "One of the first r pigeons is in hole s." This is just $P(1,s) \vee \ldots \vee P(r,s)$, but it can also be defined by recursion on r just as $R(r,s)$ is defined by recursion on s.

$H(k,r,s)$
: "At least k of the first r pigeons are in the first s holes." This will be defined by recursion on k and r. Note that $\neg\mathrm{PHP}(n)$ can be expressed as $H(n+1,n+1,n)$. The formal definitions are:

- $\neg H(k, r, s)$ whenever $k > r$. This makes sense, because k is the number of pigeons to place in the first s holes and r is the maximum number of pigeons available.
- $H(0, r, s)$, vacuously. There are always "at least zero" of anything anywhere.
- $H(k, r, s) \longleftrightarrow H(k, r-1, s) \vee [H(k-1, r-1, s) \wedge R(r, s)]$, whenever k is between 1 and r, inclusive. Either all of the k pigeons to be placed in the first s holes, or all but one of them, must be taken from the first $r-1$ pigeons; in the latter case, the remaining one can be pigeon r.

$L(k, r, s)$

This variable is defined to be equivalent to the lemma:

$H(k, r, s) \longrightarrow H(k, r, s-1) \vee [H(k-1, r, s-1) \wedge Q(r, s)]$.

If $H(k, r, s)$, then either all k pigeons, or all but one of them, are in the first $s-1$ holes; in the latter case, there must also be a pigeon in hole s. This lemma will be proved outright; the definition of $L(k, r, s)$ is just a trick to get the clauses of each case of this lemma added to the formula explicitly so that they can be reused freely without repeating the proof each time, as might be necessary if the truth of the lemma were left implicit. The purpose of the lemma is to allow (metalogical) induction on k and s when necessary, and not just on k and r. We need to be able to do case analysis on the holes a given set of pigeons are in, not just on the pigeons that are in a given set of holes.

The proof proceeds as follows. First, the lemma is established by splitting on $L(k, r, s)$ and quickly refuting the false branch each time. Since the proof works by (metalogical) induction on k and r, the splits must be done in the appropriate order, from smaller to larger values of these indices. Then we split on all cases of $H(k, r, s)$ with $k > s$: the cases that contradict the pigeonhole principle. The metalogical induction is on k and s, so again, we proceed from smaller to larger values of these indices. The lemma is essential to allow the proof to be carried out in this order. Eventually we split on $H(n+1, n+1, n)$. The true branch is refuted just as for earlier cases of $H(k, r, s)$ with $k > s$, and the false branch is refuted by showing that the given clauses of $\neg\mathrm{PHP}(n)$ entail $H(n+1, n+1, n)$, thus completing the proof.

Two important facts should be noted about this proof.

First, the DPLL tree is very deep and very lopsided; in fact, it looks more like a proof in a Frege system than like a typical DPLL tree. There is a single very long branch, representing the main line of reasoning, along which a large collection of results is accumulated, with short side branches that use proof by contradiction to establish the results one at a time, relying heavily on previously established results.

Second, although each individual definition is quite short, many of the defined variables represent formulas that would be very large and complex if expressed in clausal form directly in terms of the primitive variables $P(i, j)$. Indeed, because of the branching recursion in the definitions, the size of a straightforward rendering of $H(k, r, s)$ would grow exponentially. We are reasoning with formulas that we could not even write down in a reasonable amount of space if we were not using the extension rule!

For both of these reasons, I do not think bottom-up methods like those discussed in the previous two subsections would easily find this proof. It cannot be ruled out entirely; perhaps graph-theoretic analysis of the large 2SAT subproblems involved could find the right concepts. But it would require

very good heuristic guidance indeed to go so deep and yet branch so little as to find this proof without wasting a prohibitive amount of time on dead ends.

This is where the top-down technique of deferred splitting might be helpful. Suppose we do lookahead and select a primitive variable at random (assume without loss of generality that it is $P(n+1, n)$, since the problem is completely symmetrical at the start) and do a trial split on it. On the true branch, the subproblem that remains after unit propagation and pure variable elimination is precisely $\neg\mathrm{PHP}(n-1)$, which we want to define as $H(n, n, n-1)$. This formula shares considerable common structure with the one that we get on the false branch as well. (In fact, all the clauses of $\neg\mathrm{PHP}(n-1)$ are subsets of the clauses of the formula that remains on the false branch.)

If we were to commit immediately to this split in the fashion of straight DPLL with lookahead, we would be duplicating a great deal of the effort involved in refuting this common substructure. Instead, we can limit branching to the necessary minimum by a tactic reminiscent of dynamic programming or memoization. We abstract out the common substructure as a subproblem and try to refute it first. Only after that has been done do we actually carry out the split. Since this procedure can be carried out recursively and subproblems created deep in the deferred-split tree might be used repeatedly, it is also a good idea to define a variable equivalent to the subproblem being worked on.

On this problem, we wind up recursing all the way down to $\neg\mathrm{PHP}(2)$ before we get a trivial subproblem that collapses immediately into contradiction when we try to use lookahead to analyze further substructure. Because of unit propagation we never deal explicitly with $\neg\mathrm{PHP}(1)$.

In the current state of the art, however, this deferred splitting tactic does not quite find enough common substructure. We really need more than $\neg\mathrm{PHP}(n)$ for various n to make the proof go through; we need $H(k, r, s)$ or something very similar, and we need it for cases where k, the number of pigeons we are asserting to be in the first s holes, is considerably less than r, the number of pigeons available for placement. When, for instance, $r = 2k$, the number of ways to select pigeons for placement, which is $\mathrm{C}(2k, k)$, grows exponentially with k. The main connective in the definition of $H(k, r, s)$ is disjunction, so it can abstract from the specific choice of pigeons, which is the key to keeping the number of clauses in a proof of the pigeonhole principle reasonably low. But so far I have not been able to get the deferred splitting tactic to create subproblems whose definition has disjunction as the main connective (at least in any way that is not completely ad hoc and would have a reasonable chance of generalizing to other types of problems). The subproblems created depend too strongly on the clausal form of the whole problem, and are too fine-grained. They select specific pigeons to place in a range of holes, and we get a combinatorial explosion in the number of subproblems and definitions.

The deferred splitting idea was developed quite recently, however, and it may be that some tactic of iterating the process of finding common substructure and merging definitions disjunctively will overcome the difficulty we see in the current freeze-frame snapshot of the development of deferred splitting. I am confident that some refinement of top-down deferred splitting and subproblem creation is the most plausible path to efficient automatic generation of a polynomial-size proof of the pigeonhole principle, and this approach remains my top research priority.

6 Bibliography

6.1 Abbreviations

AAAI
American Association for Artificial Intelligence

AAAI-92
Proceedings: Tenth National Conference on Artificial Intelligence, 12-16 July, San Jose, California, USA. Sponsored by AAAI. Published by MIT Press.

AAAI-94
Proceedings of the Twelfth National Conference on Artificial Intelligence, Seattle, Washington, USA, 31 July to 4 August 1994. Two volumes. Sponsored by AAAI. Published by MIT Press.

ACM
Association for Computing Machinery

CACM
Communications of the Association for Computing Machinery

DIMACS
Center for Discrete Mathematics and Theoretical Computer Science: An NSF Science and Technology Center. Rutgers University, New Brunswick, New Jersey, USA.

IEEE
Institute of Electrical and Electronics Engineers

IJCAI-95
Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, Montreal, Quebec.

JACM
Journal of the Association for Computing Machinery

JAIR
Journal of Artificial Intelligence Research

JSL
Journal of Symbolic Logic

KR-94
Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference (KR '94), Bonn, Germany, 24-27 May 1994. Edited by Jon Doyle, E. Sandewall, and P. Torasso. Published by Morgan Kaufmann.

SIGACT
ACM Special Interest Group for Automata and Computability Theory

STOC-71

Proceedings of the Third Annual ACM Symposium on Theory of Computing, Shaker Heights, Ohio, USA, 3-5 May 1971. Sponsored by SIGACT.

WJCC-57

Proceedings of the Western Joint Computer Conference, 1957.

6.2 References

6.2.1 Books

(Feigenbaum and Feldman, 1963)

Computers and Thought. Edited by Edward A. Feigenbaum and Julian Feldman.

(Krajicek, 1995)

Bounded Arithmetic, Propositional Logic, and Complexity Theory. Jan Krajicek.

(Siekmann and Wrightson, 1983)

Automation of Reasoning. Two volumes. Edited by Jörg Siekmann and Graham Wrightson.

(van Heijenoort, 1967)

From Frege to Gödel: *A Source Book in Mathematical Logic, 1879-1931.* Jean van Heijenoort.

6.2.2 Theses, technical reports, etc.

(Blake, 1937)

Canonical Expressions in Boolean Algebra. PhD dissertation, University of Chicago, Illinois, USA.

(Cook and Mitchell, 1996)

Finding hard instances of the satisfiability problem: a survey. Stephen A. Cook and David G. Mitchell. To be published in the DIMACS series in Discrete Mathematics and Theoretical Computer Science.

(Goldberg, 1979)

On the complexity of the satisfiability problem. Courant Computer Science Report No. NSO-16, Courant Institute of Mathematical Sciences, Computer Science Department, New York University. October 1979. Allen T. Goldberg.

(Larrabee and Tsuji, 1992)

Evidence for a satisfiability threshold for random 3CNF formulas. Technical Report UCSC-CRL-92-42, CRL, University of California, Santa Cruz, November 6, 1992. Tracy Larrabee and Yumi Tsuji.

(Pretolani, 1993)

Solving satisfiability problems: an algorithm implementation challenge? Extended abstract. Workshop notes, 2nd DIMACS challenge. D. Pretolani.

(Selman, 1995)
Stochastic search and phase transitions: AI meets physics. Slides (in compressed PostScript format) for an invited talk for *IJCAI-95*. Bart Selman.

6.2.3 Articles

(Aspvall et al, 1979)
A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters* 8(3):121-123. March 1979. Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan.

(Baker, 1994)
The hazards of fancy backtracking. In *AAAI-94*, pages 288-293. Andrew B. Baker.

(Ben-Ari, 1980)
A simplified proof that regular resolution is exponential. *Information Processing Letters* 10(2):96-98. Mordechai Ben-Ari.

(Buss, 1987)
Polynomial size proofs of the propositional pigeonhole principle. *JSL* 52(4):916-927. December 1987. Samuel R. Buss.

(Chao and Franco, 1986)
Probabilistic analysis of two heuristics for the 3-satisfiability problem. *SIAM Journal on Computing* 15(4):1106-1118. Ming-te Chao and John Franco.

(Chao and Franco, 1990)
Probabilistic analysis of a generalization of the unit-clause literal selection heuristics for the k satisfiability problem. *Information Sciences* 51(3):289-314. Ming-te Chao and John Franco.

(Chvátal and Szemerédi, 1988)
Many hard examples for resolution. *JACM* 35(4):759-768. Vašek Chvátal and Endre Szemerédi.

(Cook, 1971)
The complexity of theorem-proving procedures. In *STOC-71*, pages 151-158. Stephen A. Cook.

(Cook, 1976)
A short proof of the pigeon-hole principle using extended resolution. *ACM SIGACT News* 8:28-32. October to December 1976. Stephen A. Cook.

(Cook and Reckhow, 1979)
The relative efficiency of propositional proof systems. *JSL* 44(1):36-50. March 1979. Stephen A. Cook and Robert A. Reckhow.

(Crawford and Auton, 1996)
Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence* 81(1-2):31-57. James M. Crawford and Larry D. Auton.

(Crawford and Baker, 1994)

Experimental results on the application of satisfiability algorithms to scheduling problems. In *AAAI-94*, volume 2, pages 1092-1097. James M. Crawford and Andrew B. Baker.

(Davis and Putnam, 1960)

A computing procedure for quantification theory. *JACM* 7(3):201-215. Martin Davis and Hilary Putnam. Reprinted in (Siekmann and Wrightson, 1983), volume 1, pages 125-139.

(Davis et al, 1962)

A machine program for theorem proving. *CACM* 5(7):394-397. Martin Davis, G. Logemann, and Donald W. Loveland. Reprinted in (Siekmann and Wrightson, 1983), volume 1, pages 267-270.

(Franco and Paull, 1983)

Probabilistic analysis of the Davis Putnam procedure for solving the satisfiability problem. *Discrete Applied Mathematics* 5(1):77-87. John Franco and Marvin Paull.

(Franco, 1986)

On the probabilistic performance of algorithms for the satisfiability problem. *Information Processing Letters* 23:103-106. 20 August 1986. John Franco.

(Frege, 1879)

Begriffsschrift, a *formula language, modeled upon that of arithmetic, for pure thought.* In (van Heijenoort, 1967), pages 1-82. Gottlob Frege. Translated from the German *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens.*

(Galil, 1977)

On the complexity of regular resolution and the Davis-Putnam procedure. *Theoretical Computer Science* 4(1):23-46. Zvi Galil.

(Ginsberg, 1993)

Dynamic backtracking. *JAIR* 1:25-46. Matthew L. Ginsberg.

(Ginsberg and McAllester, 1994)

GSAT and dynamic backtracking. In *KR-94*, pages 226-237. Matthew L. Ginsberg and David A. McAllester.

(Haken, 1985)

The intractability of resolution. *Theoretical Computer Science* 39:297-308. Armin Haken.

(Hogg et al, 1996)

Phase transitions and the search problem. *Artificial Intelligence* 81(1-2):1-15. Tad Hogg, Bernardo A. Huberman, and Colin P. Williams. Guest editorial for *Artificial Intelligence* "Special Volume on Frontiers in Problem Solving: Phase Transitions and Complexity."

(Jackson, 1992)

Proving unsatisfiability for problems with constant cubic sparsity. *Artificial Intelligence* 57(1):125-137. Philip C. Jackson, Jr.

(Jeroslow and Wang, 1990)

Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence* 1:167-187. Robert G. Jeroslow and Jinchang Wang.

(Konolige, 1994)

Easy to be hard: difficult problems for greedy algorithms. In *KR-94*, pages 374-378. Kurt Konolige.

(Larrabee, 1992)

Test pattern generation using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design* 11(1): 6-22. January 1992. Tracy Larrabee.

(Newell et al, 1957)

Empirical explorations with the Logic Theory Machine: a case study in heuristics. In *WJCC-57*, pages 218-239. Allen Newell, J. C. Shaw, and Herbert A. Simon. Reprinted in (Feigenbaum and Feldman, 1963), pages 109-133, and in (Siekmann and Wrightson, 1983), volume 1, pages 49-73.

(Robinson, 1965)

A machine-oriented logic based on the resolution principle. *JACM* 12(1). J. A. Robinson. Reprinted in (Siekmann and Wrightson, 1983), volume 1, pages 397-415.

(Selman et al, 1992)

A new method for solving hard satisfiability problems. In *AAAI-92*, pages 440-446. Bart Selman, Hector Levesque, and David G. Mitchell.

(Selman et al, 1994)

Noise strategies for improving local search. In *AAAI-94*, pages 337-343. Bart Selman, Henry A. Kautz, and Bram Cohen.

(Stallman and Sussman, 1977)

Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence* 9(2):135-196. Richard M. Stallman and Gerald Jay Sussman.

(Tseitin, 1968)

On the complexity of derivations in the propositional calculus. G. S. Tseitin. In *Studies in Constructive Mathematics and Mathematical Logic*, Part 2, Consultants Bureau, New York-London, pages 115-125. Reprinted in (Siekmann and Wrightson, 1983), volume 2, pages 466-483. Note: citations of the original publication in the literature are obscure and contradictory; they can't even agree on the year of publication! The version given here is from (Urquhart, 1987). Some bibliographies, including (Siekmann and Wrightson, 1983), volume 2, page 634, give the year as 1970. In any case, according to a footnote on page 466 of (Siekmann and Wrightson, 1983), volume 2, the material of this article was presented in a talk given at Leningrad in 1966.

(Urquhart, 1987)

Hard examples for resolution. *JACM* 34(1):209-219. Alasdair Urquhart.