Limit Crossing for Decision Problems

Sharlee Climer and Weixiong Zhang
Department of Computer Science and Engineering
Washington University
One Brookings Drive; St. Louis, MO 63130-4899
email: {sclimer,zhang}@cse.wustl.edu

Abstract

Limit crossing is a methodology in which modified versions of a problem are solved and compared, yielding useful information about the original problem [6, 7]. Pruning rules that are used to exclude portions of search trees are excellent examples of the limit-crossing technique. In our previous work, we examined limit crossing for optimization problems. In this paper, we extend this methodology to decision problems. We demonstrate the use of limit crossing in our design of a tool for identifying K-SAT backbones. This tool is guaranteed to identify all of the backbone variables by solving at most $n + 1$ formulae, where n is the total number of variables. While previous 3-SAT backbone research was limited to 25 variables [12] and 28 variables [9], we have computed backbones for 200 variables. In addition to being useful for identifying backbones, this code can be used directly to solve a special class of QBF problems. The code for this backbone identifier is publicly available [5].

1 Introduction

In [6, 7] the term *limit crossing* is used to refer to a general methodology that appears in a variety of guises. In essence, limit crossing is a technique in which modified versions of a problem are solved and compared, yielding useful information about the original problem. A prevalent instantiation of this procedure is the pruning method used to remove entire subtrees within a given search tree. In [7], limit crossing is examined for NP-hard combinatorial optimization problems. In [6], limit crossing is extended to general optimization problems with applications for counting problems. In this paper, we broaden the scope of limit crossing to include decision problems. We find that only a single modified version of a problem needs to be solved, and its Boolean solution is compared with the two possible

Boolean solutions of the original decision problem. This extension of limit crossing is detailed in the next section.

In section 3, we introduce a tool that utilizes decision-problem limit crossing. The decision problem of interest here is K-SAT, the NP-complete problem of determining whether a conjunctive set of disjunctive clauses, each with at most K Boolean variables, is satisfiable. Our tool identifies the *backbone* of K-SAT formulae. (A variable is part of the backbone if and only if its assignment is frozen for all valid models [9].) Moreover, we demonstrate how this tool can be used to solve a special class of Quantified Boolean Formulae (QBF). QBFs are similar to K-SAT as they are propositional formulae, however, QBFs allow the use of universal and existential quantifiers. We show experimental results on the backbone sizes of different random 3-SAT problem instances with various clause-to-variable ratios and problem sizes ranging up to 200 variables.

2 Limit Crossing for Decision Problems

In [6, 7], we discussed limit crossing for optimization problems. Within that realm, limit crossing is achieved by solving two modified versions of the original problem and comparing their solutions. We now illustrate this concept using a simple search strategy for optimally solving Max-SAT.

Max-SAT is the optimization version of K-SAT. Given n variables and m clauses containing disjunctions of variables or their negations (literals), an optimal solution for Max-SAT is an assignment of `true` or `false` to each variable such that the maximum number of clauses have a value of `true`. This maximization problem can be converted to a minimization problem, in which the number of unsatisfied clauses is minimized.

One way to solve this problem is to find an approximate solution and record the number of unsatisfied clauses, u_{ub}. u_{ub} is a upper bound on the number of unsatisfied clauses in the optimal solution. Next, we construct a binary search tree in which a variable is selected at each internal node. Consider an arbitrary node i and its associated variable, v_i. One of node i's children adds the constraint that the variable v_i is `true` and the other child requires that v_i is `false`. The children also inherit all of the assignments of their parent. For node i, the number of clauses, u_i, that are unsatisfied due to the assignments is a lower bound on the number of unsatisfied clauses for all of node i's descendants. If, at any time, $u_i \geq u_{ub}$, then node i and all of its descendants cannot lead to a better solution than the approximate solution, therefore, it can be pruned.

In this example, the first modified problem that we solve is the relaxation of the minimization requirement, yielding a simple upper bound of u_{ub}. Assigning values to variables is also an upper-bounding modification, as it cannot lead to less clauses being unsatisfied than the number that are unsatisfied in the optimal solution. However, counting the number of unsatisfied clauses entailed by the assignments associated with a given node is clearly a lower-bounding modification of the original problem. So our second modified problem is a doubly-modified problem with upper-bounding and lower-bounding components. We compare the solutions of these two modified problems at each node in the search tree. When the solution of the doubly-modified problem equals or exceeds the value of the simple upper bound, we can conclude that the upper-bounding component of the doubly-modified problem cannot lead to a better solution than our simple upper bound. That is, the assignments associated with that node are of no use.

As detailed in [6], there are many variants of limit crossing. A simple lower bound could be compared with a doubly-modified problem, and some of the upper- and/or lower-bounding modifications could be exact. However, when we consider decision problems, upper-bounding and lower-bounding modifications have no meaning. While optimization problems have numerical optimal solutions, decision problems have Boolean solutions. Nevertheless, as seen in previous work as well as the tool presented in this paper, the general concept of limit crossing is of great value in this domain.

For optimization problems, we require a simple upper bound or a simple lower bound in order to make comparisons. In the domain of decision problems, we do not need these numerical comparisons. For this reason, decision-problem limit crossing requires finding the Boolean solution of only one modified problem and comparing it with the two possible solutions of the original problem.

Two useful categories of decision-problem modifications are: *tightening* modifications and *relaxing* modifications. We define a tightened decision problem as one that can only take on the value of the original decision problem or `false` and a relaxed decision problem as one that can only take on the value of the original problem or `true`.

To illustrate limit crossing for decision problems, we consider K-SAT, the decision variant of Max-SAT. We can tighten a K-SAT instance by adding clauses, removing literals from existing clauses, and/or assigning values to variables. We can relax instances by disregarding clauses, adding literals to existing clauses, and/or allowing variables to assume both values.

If the solution of a tightened problem is `true`, we can conclude that the original problem is also `true`. Likewise, if a relaxed problem is `false`, the original problem must also be `false`.

We can also gather useful information for the other cases. For example, if a tightened problem is `false`, we can conclude that if the original problem is `true`, then the tightening modifications are invalid. This is the concept used for pruning in the Davis-Putnam algorithm for solving K-SAT problems. In this algorithm, a search tree that is similar to the tree in the Max-SAT example, is explored. Whenever the assignments associated with a node yield an unsatisfiable clause, the node is pruned. The assignments are a tightening modification, therefore they must be invalid if the original problem is `true`.

Pruning is an excellent example of limit crossing via tightening. Relaxations have also been utilized in previous work. For instance, clauses are sometimes disregarded and subformulae are solved (for example [1, 2]).

In the next section, we utilize limit crossing in our design of a backbone identifier for K-SAT.

3 Applications

In this section, we first describe our algorithm for identifying K-SAT backbones. Then we demonstrate how our backbone algorithm can be used to solve a special class of QBF problems. We close this section with backbone data that we have computed using this algorithm.

3.1 Identifying K-SAT backbones

Random K-SAT problems have been shown to exhibit characteristic easy-hard-easy *phase transitions* [4, 8]. That is, the typical-case behavior of the difficulty in solving K-SAT instances is dependent on α, the ratio of the number of clauses m to the number of variables n. Below a critical ratio $\alpha_c(K)$, almost all of the randomly generated formulae are satisfiable and finding a satisfying assignment is relatively easy. Problems with ratios that are greater than $\alpha_c(K)$ are almost all unsatisfiable and it is relatively easy to determine this unsatisfiability. However, for ratios that are near $\alpha_c(K)$, it is relatively difficult to solve K-SAT instances.

The order parameter of these phase transitions is the size of the backbone [9]. In the region very close to $\alpha_c(K)$, the backbone fraction increases from zero to one very abruptly.

Previous Max-SAT backbone research was limited to 25 variables [12] and K-SAT backbone research was limited to 28 variables [9], due to the computational demands of exhaustively enumerating all satisfied models. In this section, we present a limit-crossing K-SAT backbone identifier that requires solving $n + 1$ formulae at most. For this reason, our algorithm is beneficial when there are a large number of satisfied models and enumerating all of them is computationally demanding. (This algorithm could be modified slightly and used to find Max-SAT backbones, in which case it would require finding $n + 1$ solutions.)

The algorithm is simple. First, a given formula is solved to yield a satisfiable assignment of the variables. Then, each variable is considered, one at a time. If the value of the variable is `true` in the initial solution, then it is set to `false` or vice-versa. If the solution of this tightened problem is `false`, then we know that the tightening modification is invalid and the variable must be part of the backbone. If the tightened problem is satisfiable, we know that the variable is not part of the backbone, as we have another valid model that does not assign the same value to the variable as the initial solution. Using this new model, we can further check to see if other variables appear with different assignments and delete them from the list of candidate backbone variables. Thus we need to solve at most $n+1$ formulae.

Although many counting/enumeration problems of this sort are #P, this algorithm proves that counting/enumerating K-SAT backbone variables is NP-complete.

3.2 Application for a special class of QBF problems

Quantified Boolean Formulae (QBF) extends K-SAT by allowing the use of quantifiers. A QBF has the form:

$$Q_1 x_1 \cdots Q_n x_n E(X), X = \{x_1, \ldots, x_n\} \tag{1}$$

where each Q_i is either an existential quantifier $\exists$, or a universal quantifier $\forall$, each x_i is a Boolean variable, and E is a propositional formula. We only consider QBFs in which the propositional formula E (without any quantifiers) has at least one satisfiable assignment. QBF is a prototypical problem for the PSPACE complexity class [3].

A special QBF class has the following form:

$$\forall x_i \exists X_j E(X), x_i \in X, X_j \subseteq X, X = \{x_1, \ldots, x_n\}. \tag{2}$$

This class is of interest as

$$\forall x_i \exists X_j E(X) \Leftrightarrow x_i \notin B_E, x_i \in X, X_j \subseteq X, X = \{x_1, \ldots, x_n\} \tag{3}$$

where B_E is the set of backbone variables of E. It is clear that if this QBF is true for all values of x_i, then x_i is not a backbone variable. Furthermore, if x_i is not part of the backbone, then there must exist at least one valid model of E in which x_i is `false`, and at least one in which it is `true`.

Equivalence (3) allows us to use our backbone identifier to solve QBFs that have the form of (2).

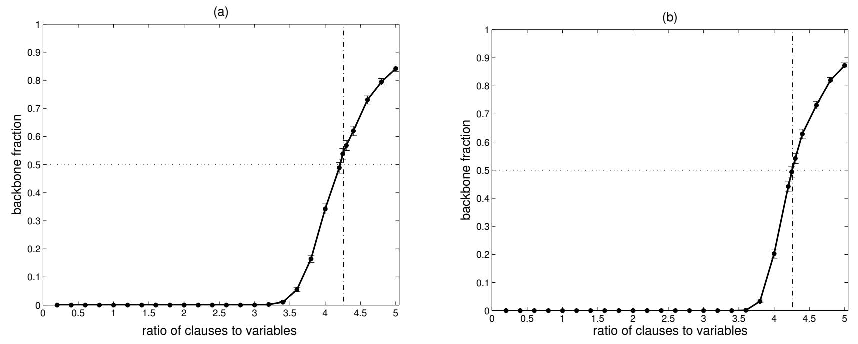

Figure 1: Average 3-SAT backbone fractions as a function of the ratio of clauses to variables, α. The vertical line marks $\alpha = 4.26 \approx \alpha_c(3)$. The 95% confidence interval is shown by bars. (a) 50 variables. (b) 100 variables.

3.3 Experimental results on K-SAT backbone

We implemented our backbone finder in C++. For solving the formulae, we used zChaff [11], the award-winning publicly-available K-SAT solver that was developed at Princeton. We added the backbone-finding code on top of zChaff. Our backbone-finder code will be made available [5].

Using this backbone identifier, we computed backbones for random 3-SAT formulae. We constructed the formulae by randomly selecting three variables per clause, such that no variable appears twice within a single clause. We then negated variables with a probability of 0.5. We varied α, the ratio of the number of clauses to the number of variables, and computed the fraction of variables that are part of the backbone. Each data point in the following graphs represents the mean or median of 1,000 satisfiable trials. Finally, the data presented here was computed in a short period of time. The final version of this paper will contain more extensive results for 3-SAT, as well as backbones for 4-SAT, 5-SAT, and 6-SAT.

4 Conclusions

In Pearl's book *Heuristics: Intelligent Search Strategies for Computer Problem Solving* [10], he discusses automatically generating admissible and consistent heuristic functions by systematically relaxing constraints entailed by a given problem. He points out that this approach can lead to identifying heuristics that might otherwise be overlooked by pure "discovery". To illustrate this point, he cites such an occurrence for heuristic functions for the 8-Puzzle problem.

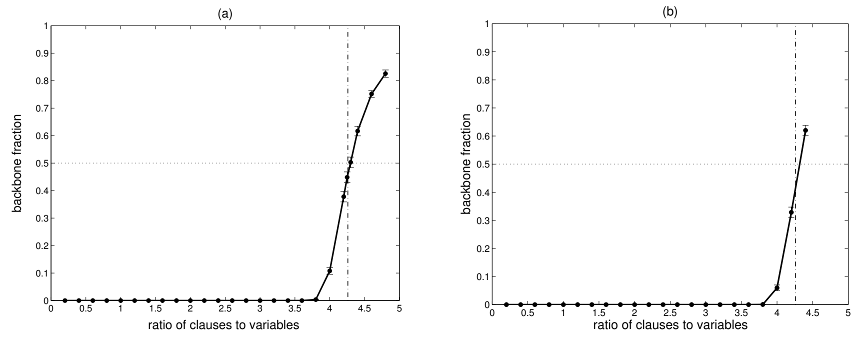

Figure 2: Average 3-SAT backbone fractions as a function of the ratio of clauses to variables, α. The vertical line marks $\alpha = 4.26 \approx \alpha_c(3)$. The 95% confidence interval is shown by bars. (a) 150 variables. (b) 200 variables.

Limit crossing is an intuitive concept and is the basis of many "discoveries" in previous work. However, by developing a more systematic approach for utilizing this concept, we may be able to devise algorithms that may have otherwise been overlooked. For example, the K-SAT backbone identifier presented in this paper is an obvious idea when approached with a limit-crossing point of view, however, it has been previously overlooked, resulting is severe limitations for empirical K-SAT backbone research.

Although the intuitive concept has been around for a period of time, a generalized model of this technique has not. In this, and our previous work, we have attempted to build such a model and identify potential modifications of problems that may be of use. At the same time, we have developed limit-crossing algorithms that proven to be effective, especially for difficult problems. For our future work, we plan to continue exploiting this powerful methodology.

Acknowledgment

This research was supported in part by NSF grants IIS-0196057 and ITR/EIA-0113618, NDSEG and Olin Fellowships, and in part by DARPA Cooperative Agreements F30602-00-2-0531 and F33615-01-C-1897. Thanks to Guandong Wang for helpful discussions.

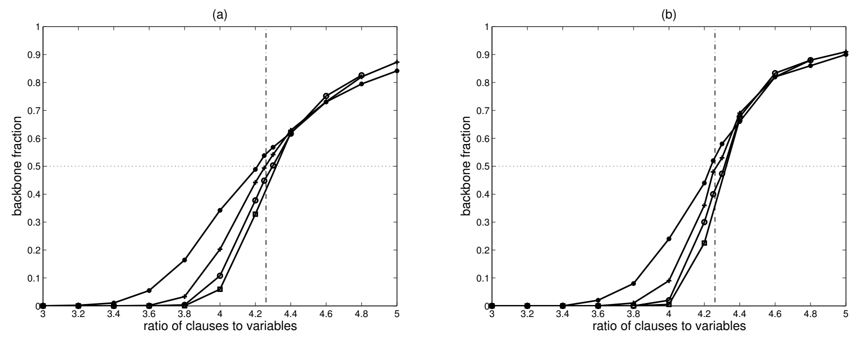

Figure 3: (a) Average values for 50, 100, 150, and 200 variables. (b) Median values for 50, 100, 150, and 200 variables.

References

[1] E. Amir and S. McIlraith. Solving satisfiability using decomposition and the most constrained subproblem (preliminary report). In *Proceedings of LICS 2001 Workshop on Theory and Applications of Satisfiability Testing (SAT 2001)*, Boston, Massachusetts, June 2001.

[2] R. Bruni and A. Sassano. Restoring satisfiability or maintaining unsatisfiability by finding *small* unsatisfiable subformulae. In *Proceedings of LICS 2001 Workshop on Theory and Applications of Satisfiability Testing (SAT 2001)*, Boston, Massachusetts, June 2001.

[3] M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. An algorithm to evaluate quantified boolean formulae and its experimental evaluation. In *Journal of Automated Reasoning*, June 1999.

[4] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence, (IJCAI-91)*, pages 331–337, Sydney, Australia, August 1991.

[5] S. Climer. k-sat backbone identifier code. Forthcoming.

[6] S. Climer and W. Zhang. Limit crossing methodology and its application to counting problems. Submitted to IJCAI-03.

[7] S. Climer and W. Zhang. Searching for backbones and fat: A limit-crossing approach with applications. In *Proceedings of the 18th National Conference*

on Artificial Intelligence (AAAI-02), pages 707–712, Edmonton, Alberta, July 2002.

[8] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92)*, pages 459–465, San Jose, CA, July 1992.

[9] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic ‘phase transitions’. *Nature*, 400:133–137, 1999.

[10] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading, MA, 1984.

[11] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of ICCAD 2001*, San Jose, CA, November 2001.

[12] W. Zhang. Phase transitions and backbones of 3-sat and maximum 3-sat. In *7th Intern. Conf. on Principles and Practice of Constraint Programming (CP-2001)*, pages 153–167, 2001.