How much backtracking does it take to color random graphs? Rigorous results on heavy tails

Haixia Jia and Cristopher Moore

Computer Science Department, University of New Mexico, Albuquerque NM 87131
{hjia,moore}@cs.unm.edu

Abstract. For many backtracking search algorithms, the running time has been found experimentally to have a heavy-tailed distribution, in which it is often much greater than its median. We analyze two natural variants of the Davis-Putnam-Logemann-Loveland (DPLL) algorithm for Graph 3-Coloring on sparse random graphs of average degree c. Let $P_c(b)$ be the probability that DPLL backtracks b times. First, we calculate analytically the probability $P_c(0)$ that these algorithms find a 3-coloring with no backtracking at all, and show that it goes to zero faster than any analytic function as $c \to c^* = 3.847...$ Then we show that even in the "easy" regime $1 < c < c^*$ where $P_c(0) > 0$ — including just above the degree $c = 1$ where the giant component first appears — the expected number of backtrackings is exponentially large with positive probability. To our knowledge this is the first rigorous proof that the running time of a natural backtracking algorithm has a heavy tail for graph coloring.

1 Introduction

Many common search algorithms for combinatorial problems have been found to exhibit a heavy-tailed distribution in their running times; for instance, in the number of backtracks performed by the Davis-Putnam-Logemann-Loveland (DPLL) algorithm and its variants on constraint satisfaction problems, including Satisfiability, Graph Coloring, and Quasigroup Completion [7–10, 12]. In such a distribution, with significant probability, the running time is much larger than its median, and indeed its expectation can be exponentially large even if the median is only polynomial. These distributions typically take a power-law form, in which the probability $P(b)$ that the algorithm backtracks b times behaves as $b^{-\gamma}$ for some exponent γ. One consequence of this is that if a run of the algorithm has taken longer than expected, it is likely to take much longer still, and it would be a good idea to restart it (and follow a new random branch of the tree) rather than continuing to search in the same part of the search space.

For Graph 3-Coloring, in particular, these heavy tails were found experimentally by Hogg and Williams [10] and Davenport and Tsang [5]. At first, it was thought that this heavy tail indicated that many *instances* are exceptionally hard. A clearer picture emerged when Gomes, Selman and Crato [8] showed that the running times of randomized search algorithms on a typical *fixed* instance show a heavy tail. In Figure 1 we show our own experimental data on

the distribution of the number of backtracks for two versions of DPLL described below. In both cases the log-log plot follows a straight line, indicating a power law. As n increases, the slopes appear to converge to -1, and we conjecture that $P_c(b) \sim b^{-1}$ up to some exponential cutoff.

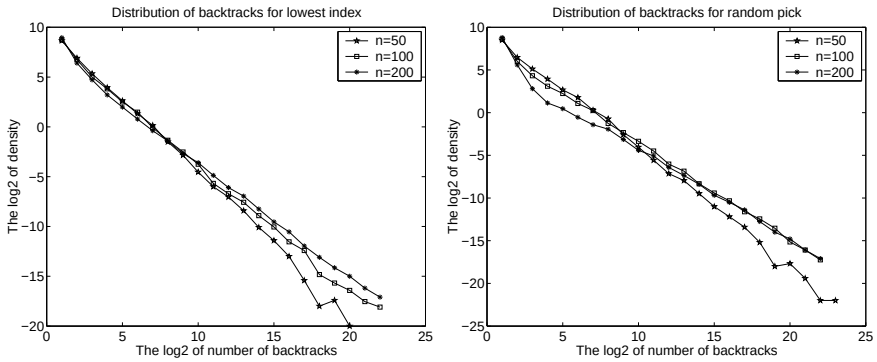

Fig. 1. Log-log plots of the distribution of the number of backtracks $P_c(b)$ for the two DPLL algorithms `A` and `B` described in the text on random graphs with $c = 3.5$. The data appears to follow a power law $P_c(b) \sim b^{-1}$ in the limit $n \to \infty$.

A fair amount of theoretical work has been done on heavy tails, including optimal restart strategies [11] and formal models [4]. However, there are relatively few rigorous results establishing that these tails exist. One exception is Achlioptas, Beame, and Molloy [1], who showed using lower bounds on resolution proof complexity that DPLL takes exponential time on random instances of 3-SAT, even at some densities below the satisfiability threshold. Our results appear to be the first on Graph Coloring, and they rely on much simpler reasoning.

Our results hold for two variants of DPLL. Both of them are greedy, in the sense that they branch on a vertex with the smallest available number of colors; in particular, they perform *unit propagation*, in which any 1-color vertex is immediately assigned that color. They are distinguished by which 2-color vertex they branch on when there are no 1-color vertices (we focus on the algorithm's performance on the giant component, during which there is always a 1- or 2-color vertex). In algorithm `A`, the vertices are given a fixed uniformly random ordering, and we branch on the 2-color vertex of lowest index. In algorithm `B`, we choose a vertex uniformly at random from among the 2-color vertices. In both variants, we try the two possible colors of the chosen 2-color vertex in random order.

Our main result is the following:

Theorem 1. *For algorithms* `A` *and* `B`*, let b be the number of times the algorithm backtracks on $G(n, c/n)$. If $1 < c < c^* = 3.847...$, there exist constants $\beta, q > 0$ such that $\Pr[b > 2^{\beta n}] \geq q$. In particular, $\mathrm{E}[b] = 2^{\Theta(n)}$.*

Note that this theorem does not show that this tail has a power-law form (although we believe our arguments could be refined to do that); it simply shows that with positive probability the amount of backtracking is exponentially large.

We rely heavily on the fact that for both these variants of DPLL, a single random branch is equivalent to a linear-time greedy heuristic, 3-GL, analyzed by Achlioptas and Molloy [2]. They showed that if $1 < c < c^* = 3.847...$ then 3-GL colors $G(n, c/n)$ with positive probability. (If $c < 1$ then the graph w.h.p. has no bicyclic component and it is easy to color.) This shows that $P_c(0) > 0$ for c in this range, i.e., with positive probability these variants of DPLL succeed with no backtracking at all. However, as our results show, the expected amount of backtracking is exponentially large even for random graphs with c in this "easy" regime, and indeed just above the appearance of the giant component at $c = 1$.

2 The probability of success without backtracking

We follow an approach of Achlioptas and Moore [3] and separate the algorithm's progress into rounds, where each round consists of coloring a 2-color vertex and the resulting cascade of 1-color vertices. We use generating functions to calculate exactly the probability that a given round will fail (and backtrack) by creating a 0-color vertex. This gives the following result:

$$P_c(0) = \exp\left(-\int_0^{t_0} \mathrm{d}t\, \frac{c\lambda^2}{2(1-\lambda)(2+\lambda)}\right) + o(1) \ . \tag{1}$$

Here t_0 is the unique positive root of $1-t-\mathrm{e}^{-ct} = 0$ and $\lambda = (2/3)c(1-t-\mathrm{e}^{-ct})$. Figure 2 shows that (1) fits our experimental data for graphs of size 10^4 perfectly.

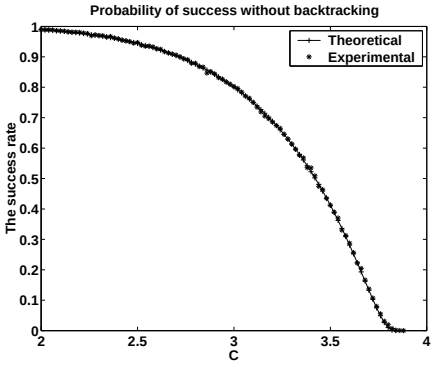

Fig. 2. A comparison of our calculation (1) of the probability of success (the solid line) with experimental results with $n = 10^4$. Each point is the average of 10^4 trials.

As c approaches the unique positive root c^* of $c - \ln c = 5/2$, the integral in (1) diverges and $P_c(0) \to 0$ faster than any analytic function. Specifically, for

$c = c^* - \epsilon$ we have $P_c(0) \sim \exp(-A/\sqrt{\epsilon})$. Using methods from statistical physics, Deroulers and Monasson [6] found the same critical behavior for heuristics on random 3-SAT; we expect it for other graph coloring heuristics as well, such as the smoothed Brelaz heuristic analyzed by Achlioptas and Moore [3] which succeeds with positive probability for $c < c^* \approx 4.03$.

3 Exponential backtracking with positive probability

In this section we sketch the proof of Theorem 1. We focus on variant `A` of DPLL; the reasoning for `B` is similar.

Let t_1 be a constant such that $0 < t_1 < t_0$ where t_0, given in the previous section, is the expected time for the algorithm to color the giant component if it does so without backtracking. Run the algorithm for $t_1 n$ steps, conditioning on not having created a 0-color vertex so far; this is equivalent to running 3-GL conditioned on its success. At the end of these $t_1 n$ steps there are w.h.p. $s_3(t_1)n + o(n)$ 3-color vertices and $s_2(t_1)/3 + o(n)$ 2-color vertices of each color pair, where $s_3(t_1)$ and $s_2(t_1)$ are given by the differential equations of [2]. In addition, the uncolored part of the graph G' is uniformly random in $G(n', p)$ where n' is the total number of uncolored vertices.

Let us call a triangle *bad* if it is composed of 2-color vertices whose allowed colors are red and green, it is disconnected from the rest of G', and the indices of its vertices are all greater than the median index of the 2-color vertices in G'. The number of bad triangles in G' is essentially Poisson distributed with expectation $\Theta(1)$, so with positive probability there is exactly one of them.

Let us call this bad triangle Δ. It is important to us in the following ways:

1. It is not 2-colorable, so every branch of the subtree below the step creating Δ will fail, and the algorithm will be forced to backtrack at least to the $(t_1 n)$th step and uncolor one of Δ's blue neighbors.
2. Since Δ is isolated from rest of G', we will find this contradiction only if we choose one of Δ's vertices from the pool of 2-color vertices. In particular, we will not be led to Δ by a chain of forced steps.
3. `A` will not color any of Δ's vertices until it runs out of 2-color vertices of lower index, and this will not happen for at least $s_2(t_1)n/2$ more steps.

Therefore, Δ will cause the entire subtree starting with these $t_1 n$ steps to fail, but we won't find out about it until we explore the tree $\Theta(n)$ more deeply, and this forces us visit an exponential number of nodes.

Overall, what we are saying is that the events that different branches of the search tree fail are far from independent; they have a strong positive correlation since a single bad triangle dooms an entire subtree to failure, and the probability they all fail is positive even though a random branch succeeds with positive probability $P_c(0)$. The DPLL algorithm naively tries to 2-color Δ an exponential number of times, hoping that recoloring other vertices will render Δ 2-colorable. In terms of restarts, once Δ has "spoiled" an entire section of the search space, it makes more sense to start over with a new random branch.

We would like to go beyond Theorem 1 and prove a power-law distribution, $P_c(b) \sim b^{-1}$. Intuitively, suppose that Δ appears at a uniformly random depth d up to a maximum of n, and that the running time b scales as 2^{Ad} for some A. Then the probability that b is between 2^{Ad} and $2^{A(d+1)}$ is $1/n$, giving a probability density $P(b) = 1/(2^{Ad}(2^A - 1)n) \sim 1/b$. Any distribution of d which varies slowly from $\Theta(1)$ to $\Theta(n)$ would give the same qualitative result.

A logical question is what happens to the running time if we backtrack immediately whenever we create an odd 2-color cycle, rather than waiting to bump into it deeper in the search. While this obviates our proof of Theorem 1, we find experimentally that this variant of DPLL still has a heavy-tailed distribution of running times. We propose this as a direction for further work.

Acknowledgments. We are grateful to Dimitris Achlioptas, Sinan Al-Saffar, Paul Beame, Tracy Conrad, Michael Molloy, Remi Monasson, Vishal Sanwalani and Bart Selman for helpful comments and conversations. This work was supported by NSF grant PHY-0200909 and the Los Alamos National Laboratory.

References

1. Dimitris Achlioptas, Paul Beame, and Michael Molloy, "A sharp threshold in proof complexity." *Proc. 33rd Symp. on Theory of Computing* 337–346.
2. D. Achlioptas and M. Molloy, "Analysis of a list-colouring algorithm on a random graph." *Proc. 38th Foundations of Computer Science* 204–212.
3. D. Achlioptas and C. Moore, "Almost all graphs of degree 4 are 3-colorable." *Proc. 34th Symp. on Theory of Computing* 199–208, and *J. Comp. & Sys. Sci.* 67 (2003) 441–471, special issue for STOC 2002.
4. H. Chen, C.P. Gomes and B. Selman, "Formal models of heavy-tailed behavior in combinatorial search." *Proc. 7th Intl. Conf. on the Principles and Practice of Constraint Programming* (2001) 408–422.
5. A. Davenport and E.P.K. Tsang, "An empirical investigation into the exceptionally hard problems." *Proc. Workshop on Constraint-based Reasoning* 46–53.
6. C. Deroulers and R. Monasson, "Critical behaviour of combinatorial search algorithms, and the unitary-propagation universality class." Preprint, cond-mat/0405319.
7. I. Gent, and T. Walsh, "Easy problems are sometimes hard." *Artificial Intelligence* 70 (1993) 335–345.
8. C.P. Gomes, B. Selman and N. Crato, "Heavy-Tailed Distributions in Combinatorial Search." *Proc. 3rd Intl. Conf. on Principles and Practices of Constraint Programming* (1997) 121–135.
9. C.P. Gomes, B. Selman and H.A. Kautz, "Boosting Combinatorial Search Through Randomization." *Proc. 15th Natl. Conf. on Artificial Intelligence* (1998) 431–437.
10. T. Hogg and C.P. Williams, "The Hardest Constraint Problems: A Double Phase Transition." *Artificial Intelligence* 69(1-2) (1994) 359–377.
11. M. Luby, A. Sinclair, and D. Zuckerman, "Optimal speedup of las vegas algorithms." *Information Processing Letters* (1993) 173–180.
12. B. Selman, H. Kautz, and B. Cohen, "Local search strategies for satisfiability testing." In D. Johnson and M. Trick, Eds., *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 26 (1993) 521–532.