Determining the expected runtime of an exact graph coloring algorithm

Zoltán Ádám Mann
Budapest University of Technology and Economics
Department of Computer Science and Information Theory
Magyar tudósok körútja 2., 1117 Budapest, Hungary
zoltan.mann@cs.bme.hu

Anikó Szajkó
Budapest University of Technology and Economics
Department of Computer Science and Information Theory
Magyar tudósok körútja 2., 1117 Budapest, Hungary
szajko.aniko@gmail.com

ABSTRACT

Exact algorithms for graph coloring tend to have high variance in their runtime, posing a significant obstacle to their practical application. The problem could be mitigated by appropriate prediction of the runtime. For this purpose, we devise an algorithm to efficiently compute the expected runtime of an exact graph coloring algorithm as a function of the parameters of the problem instance: the graph's size, edge density, and the number of available colors. Specifically, we investigate the complexity of a typical backtracking algorithm for coloring random graphs with k colors. Using the expected size of the search tree as the measure of complexity, we devise a polynomial-time algorithm for predicting algorithm complexity depending on the parameters of the problem instance. Our method also delivers the expected number of solutions (i.e., number of valid colorings) of the given problem instance, which can help us decide whether the given problem instance is likely to be feasible or not. Based on our algorithm, we also show in accordance with previous results that increasing the number of vertices of the graph does not increase the complexity beyond some complexity limit. However, this complexity limit grows rapidly when the number of colors increases.

1. INTRODUCTION AND PREVIOUS WORK

Graph coloring[1] is one of the most fundamental problems in algorithmic graph theory, with many practical applications, such as register allocation, frequency assignment, pattern matching, and scheduling [22, 6, 19]. Unfortunately, graph coloring is NP-complete [11]. Moreover, if $P \neq NP$, then no polynomial-time approximation algorithm with an approximation factor smaller than 2 can exist for graph coloring [10].

[1]See Section 2 for detailed definitions.

Exact graph coloring algorithms are often variants of the usual backtrack algorithm. The backtrack algorithm has the advantage that, by pruning large parts of the search tree, it can be significantly more efficient than checking the whole search space exhaustively. In the worst case, the backtrack algorithm requires an exponential number of steps, but its average-case complexity is $O(1)$ [27]. The runtime can vary significantly: both very short and very long runs have non-negligible probability [16].

The probabilistic analysis of the coloring of random graphs was first suggested in the seminal paper of Erdős and Rényi [9]. Subsequent work of Grimmett and McDiarmid [13], Bollobás [4], and Luczak [17], lead to an understanding of the order of magnitude of the expected chromatic number of random graphs. Through the recent work of Shamir and Spencer [24], Luczak [18], Alon and Krivelevich [2], and Achlioptas and Naor [1], we can determine almost exactly the expected chromatic number of a random graph in the limit: with probability tending to 1 when the size of the graph tends to infinity, the expected chromatic number of a random graph is one of two possible values.

In terms of the running time of the backtracking algorithm on random graphs, much less is known. Bender and Wilf gave lower and upper bounds on the runtime of backtracking in the non-k-colorable case [3]. Asymptotically, these bounds are quite good, but in practical cases, there can be several orders of magnitude difference between the lower and upper bounds. In a recent paper, we improved these bounds [20], but there still exists a range of the input parameters, in which there is a non-negligible gap between the lower and upper bounds (see Figure 1, and note also the exponential scale on the vertical axis). Hence, accurate prediction of the algorithm's runtime is still only partially possible.

Predicting the runtime of the algorithm would greatly improve its practical usability, by informing the user in advance about the estimated runtime. This would let the user decide if the exact solution of the problem is realistic in the available time frame, or a heuristic solution should be used instead. More generally, it allows the manual or automated selection of the most suitable algorithm from an algorithm portfolio [12]. It also enhances load balancing when

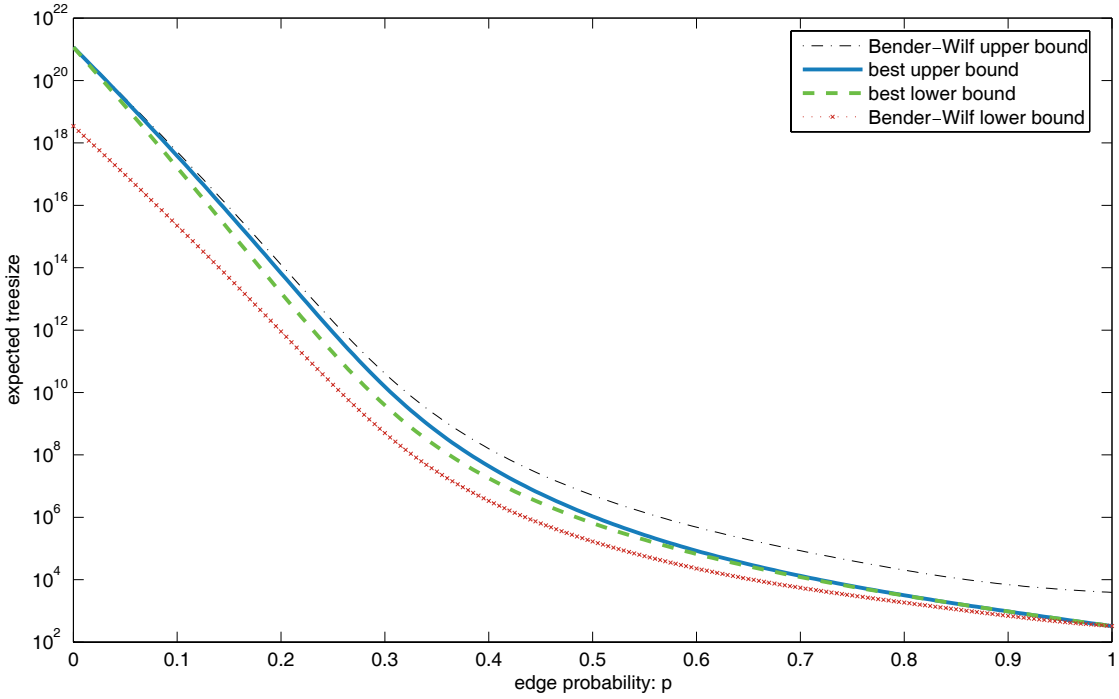

Figure 1: Lower and upper bounds on algorithm runtime of Bender and Wilf [3] and the best known bounds from our recent paper [20]

several problem instances are solved in parallel on multiple machines.

Empirical study of the behaviour of search algorithms and the complexity of graph coloring problem instances has lead to the discovery of a phase transition phenomenon with an accompanying easy-hard-easy pattern [7, 15, 14, 8]. Briefly, this means that, given k colors, for small values of the edge density (underconstrained case), almost all random graphs are colorable. When the edge density of the graph increases, the ratio of k-colorable graphs abruptly drops from almost 1 to almost 0 (phase transition). After this critical region, almost all graphs are non-k-colorable (overconstrained case). In the underconstrained case, coloring is easy: even the simplest heuristics usually find a proper coloring [26, 5]. In the overconstrained case, it is easy for backtracking algorithms to prove uncolorability because they quickly reach contradiction [23]. The hardest instances lie in the critical region [7]. This phenomenon is exemplified in Figure 2, showing our own empirical findings, experimenting with a backtrack graph coloring algorithm [21].

Summarizing these results, one can state that we have a good *quantitative* understanding of graph coloring *in the limit* (when the size of the graph tends to infinity) and a good *qualitative* understanding of it in the finite case. Our aim in this paper is to study the hardness of graph coloring *quantitatively* with accurate results for *finite* graphs.

Hence, our aim is to devise an algorithm for obtaining accurate results on the expected runtime of the backtrack algorithm in coloring random graphs. Like [3] and [20], we restrict ourselves to the non-k-colorable case (see Section 2). More specifically, our complexity results are accurate in the non-k-colorable case, but only an upper bound in the k-colorable case. To use a machine independent measure of algorithm complexity, we analyze the expected size of the search tree as a function of problem instance paramteres: the size of the graph, the edge density and the number of available colors (Section 3). Our contribution is an algorithm for determining the expected size of the search tree exactly (Section 4). The algorithm uses dynamic programming, and its runtime is polynomial in the size of the graph. As a by-product, we also obtain the exact value of the expected number of solutions as a function of input parameters (Section 5). We also present our empirical findings on how the complexity of the problem and the number of solutions depend on the input parameters (Section 6). Finally, Section 7 concludes the paper.

2. PRELIMINARIES

We consider the decision version of the graph coloring problem, in which the input consists of an undirected graph $G = (V, E)$ and a number k, and the task is to decide whether the vertices of G can be colored with k colors such that adjacent vertices are not assigned the same color. The input graph is a random graph from $G_{n,p}$, i.e. it has n vertices and each pair of vertices is connected by an edge with probability p independently from each other. The vertices of the graph will be denoted by $v_1, \ldots, v_n$, the colors by

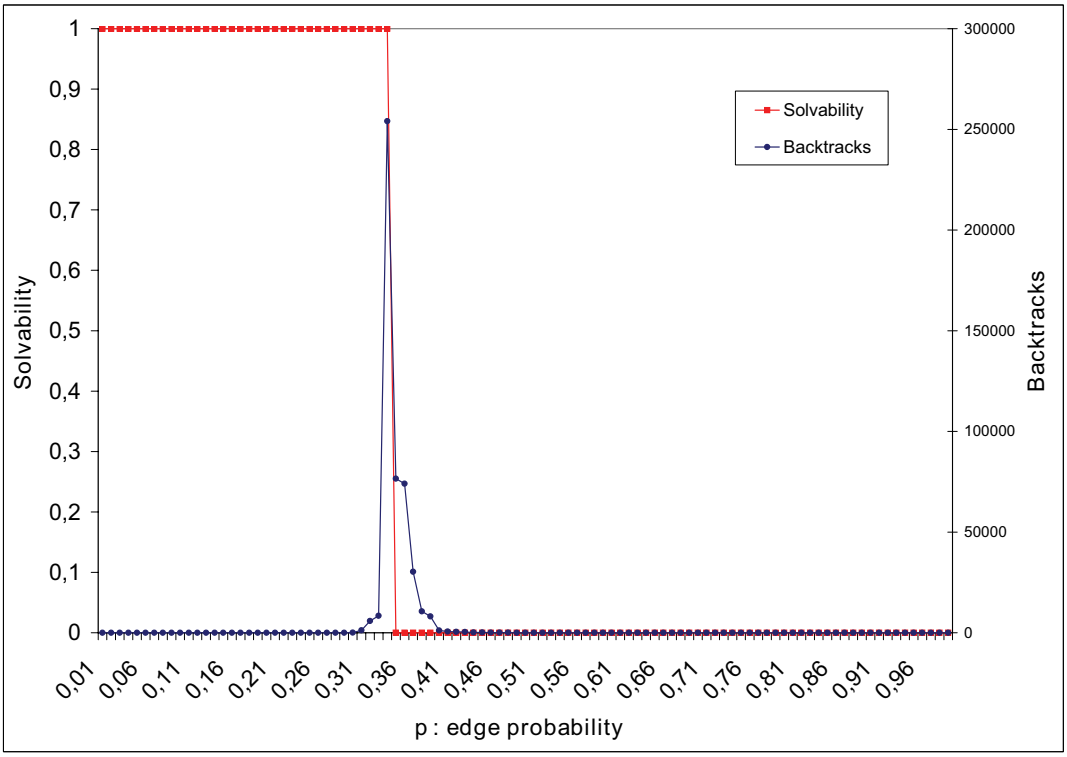

Figure 2: The runtime complexity of a backtrack algorithm (right y-axis: number of backtracks) and the ratio of feasible problem instances (left y-axis: solvability) in the coloring of $G_{n,p}$ random graphs for $n = 70$, as a function of the edge probability (p). The number of colors is $k = 8$.

$1, \ldots, k$. A *coloring* assigns a color to each vertex; a *partial coloring* assigns a color to some of the vertices. A (partial) coloring is *invalid* if there is a pair of adjacent vertices with the same color, otherwise the (partial) coloring is *valid*.

The backtrack algorithm considers partial colorings. It starts with the empty partial coloring, in which no vertex has color. This is the root – that is, the single node on level 0 – of the search tree[2]. Level t of the search tree contains the k^t possible partial colorings of $v_1, \ldots, v_t$. The search tree, denoted by T, has $n + 1$ levels $(0, 1, \ldots, n)$, the last level containing the k^n colorings of the graph. For simplicity of notation, we use $w \in T$ to denote that the partial coloring w is a node of the search tree. Furthermore, let T_t denote the set of partial colorings on level t of T. If $t < n$ and $w \in T_t$, then w has k children in the search tree: those partial colorings of $v_1, \ldots, v_{t+1}$ that assign to the first t vertices the same colors as w.

A node $w \in T_t$ is a partial coloring, i.e. it can also be regarded as a function $w : \{v_1, v_2, \ldots v_t\} \to \{1, 2, \ldots, k\}$. That is, for $w \in T_t$ and $v \in \{v_1, v_2, \ldots v_t\}$, $w(v)$ denotes the color of vertex v in the partial coloring w. In other cases, i.e. when $t < i \leq n$, then $w(v_i)$ is undefined as w assigns no color to v_i.

[2]In order to avoid misunderstandings, we use the term 'vertex' in the case of the input graph and the term 'node' in the case of the search tree.

In each partial coloring w, the backtrack algorithm considers the children of w and visits only those that are valid. Invalid children are not visited, and this way, the whole subtree under an invalid child of the current node is pruned. This is correct because all nodes in such a subtree are also certainly invalid.

T depends only on n and k, not on the specific input graph. However, the algorithm visits only a subset of the nodes of T, depending on which vertices of G are actually connected. The number of actually visited nodes of T will be used to measure the complexity of the algorithm on the given problem instance. Moreover, the number of actually visited nodes on the nth level of T yields the number of solutions.

Of course, this is a simplified algorithm model. In practice, a backtracking graph coloring algorithm can be enhanced with several techniques, e.g. heuristics for the choice of the next vertex to color and the order in which the colors should be considered, symmetry breaking, consistency propagation etc. [25]. Nevertheless, this simplified model captures well the main phenomena of any backtracking-style algorithm (i.e., branching as well as pruning invalid subtrees of the search tree), especially in the non-k-colorable case. This is because in the non-k-colorable case, the search space to be traversed is to a large extent given, and the algorithm must traverse all of it (except for the pruned subtrees of the search tree). In the k-colorable case, the ideas mentioned above for speeding up the algorithm can be leveraged more intensively.

E.g., with lucky choices of the vertices to color and the colors to assign to them, the algorithm might manage to color the graph with a very small amount of – or even zero – backtracking. In contrast, in the non-k-colorable case, the order in which the colors to assign to a given vertex are tried does not matter because all of them have to be tried anyway. Therefore, our model is realistic in the non-k-colorable case; in the k-colorable case, our complexity results can be seen as upper bounds on real algorithm complexity.

3. THE EXPECTED NUMBER OF VISITED NODES OF THE SEARCH TREE

For each $w \in T$, we define the following random variable (the value of which depends on the choice of G):

$$Y_w = \begin{cases} 1 & \text{if } w \text{ is valid,} \\ 0 & \text{else.} \end{cases}$$

Let $p_w = Pr(Y_w = 1)$. Moreover, we define one more random variable (whose value also depends on the choice of G): $Y =$ the number of visited nodes of T.

Since the algorithm visits exactly the valid partial colorings, it follows that $Y = \sum_{w \in T} Y_w$, and thus $E(Y) = \sum_{w \in T} E(Y_w)$. Moreover, it is clear that $E(Y_w) = p_w$. It follows that the expected number of visited nodes in T is:

$$E(Y) = \sum_{w \in T} p_w.$$

For $w \in T_t$, let

$$Q(w) := \big\{\{x, y\} : x, y \in \{v_1, \ldots, v_t\}, x \neq y, w(x) = w(y)\big\}$$

be the set of pairs of vertices with identical colors, and let $q(w) := |Q(w)|$. Clearly, w is valid if and only if, for all $\{x, y\} \in Q(w)$, x and y are not adjacent. It follows that $p_w = (1-p)^{q(w)}$ and thus the expected number of visited nodes of T is:

$$E(Y) = \sum_{w \in T} (1-p)^{q(w)}.$$

Note that computing $E(Y)$ directly through this formula is not tractable since $|T|$ is exponentially large in n. In the following, we devise a way to overcome this hurdle by a smart grouping of the terms of this sum.

4. EFFICIENT CALCULATION USING DYNAMIC PROGRAMMING

Before presenting our algorithm, we need to introduce some further notions. Our first aim is to compute the maximum possible value of $q(w)$ within T_t.

We denote by $s(w, i)$ (or simply s_i if it is clear which partial coloring is considered) the number of vertices of G that are assigned color i in the partial coloring w.

Proposition 1. *For all $w \in T_t$, $q(w) \leq \binom{t}{2}$.*

Proof.

$$\begin{aligned} q(w) &= \sum_{i=1}^{k} \binom{s_i}{2} = \frac{1}{2}\left(\sum_{i=1}^{k} s_i^2 - \sum_{i=1}^{k} s_i\right) \leq \\ &\leq \frac{1}{2}\left(\left(\sum_{i=1}^{k} s_i\right)^2 - \sum_{i=1}^{k} s_i\right) = \frac{1}{2}\left(t^2 - t\right) = \binom{t}{2}. \end{aligned}$$

□

We will denote this value as $q_{max}(t)$ or simply q_{max}. It is also possible to derive a formula for the minimum of $q(w)$ [20], depending on the value of t and k. This value will be denoted by $q_{min}(t, k)$ or simply q_{min}. The exact formula for $q_{min}(t, k)$ is not necessary for our purposes.

Let $R(q, t, k) := |\{w \in T_t : q(w) = q\}|$ denote the frequency of value q among the $q(w)$ values of the nodes in T_t, given k colors. (The right-hand-side of the definition of $R(q, t, k)$ does not seem to depend on k. However, w inherently depends on k.)

In the sum $\sum_{w \in T_t} (1-p)^{q(w)}$, we can group the terms according to the q values. Since $R(q, t, k)$ is the frequency of the value q among the $q(w)$ values of nodes in T_t, we obtain

$$\sum_{w \in T_t} (1-p)^{q(w)} = \sum_{q=q_{min}(t)}^{q_{max}(t)} R(q, t, k)(1-p)^q.$$

Therefore,

$$E(Y) = \sum_{w \in T} (1-p)^{q(w)} = \sum_{t=0}^{n} \sum_{q=q_{min}(t)}^{q_{max}(t)} R(q, t, k)(1-p)^q.$$

If we could determine all the $R(q, t, k)$ values explicitly, this would enable us to efficiently calculate the exact value of $E(Y)$ using this formula. Determining the $R(q, t, k)$ values is possible with the following recursion (we write ℓ instead of k as third parameter, so that the meaning of k is not affected):

Proposition 2.

$$R(q, t, \ell) = \sum_{j=0}^{t} \binom{t}{j} R\left(q - \binom{j}{2}, t - j, \ell - 1\right).$$

Proof. Assume that color class 1 contains j vertices. There are $\binom{t}{j}$ possibilities to choose these j vertices. The remaining $t - j$ vertices must be colored with $\ell - 1$ colors. Moreover, the j vertices of color 1 already account for $\binom{j}{2}$ vertex pairs with identical colors. Hence, the remaining $t - j$ vertices must be colored in such a way that the number of vertex pairs with identical colors out of these $t - j$ vertices equals $q - \binom{j}{2}$. For this, there are exactly $R\left(q - \binom{j}{2}, t - j, \ell - 1\right)$ possibilities. □

Based on this recursive formula, we can use dynamic programming to compute the $R(q, t, \ell)$ values and store them in

Algorithm 1 Dynamic programming algorithm to compute $E(Y)$

```
//Set R values for ℓ = 1
for t=0 to n
{
    R(C(t,2), t, 1) = 1
}

//Set R for higher values of ℓ
for ℓ=2 to k
{
    for t=0 to n
    {
        for q=q_min to q_max
        {
            //Use the recursive formula to compute R
            R(q,t,ℓ) = 0
            for j=0 to t
            {
                //Consider the current term only if non-zero
                if q − C(j,2) ≥ q_min(t − j, ℓ − 1)
                {
                    term = C(t,j) R(q − C(j,2), t − j, ℓ − 1)
                    R(q,t,ℓ) = R(q,t,ℓ) + term
                }
            }
        }
    }
}

//Compute E(Y)
result=0
for t=0 to n
{
    for q=q_min to q_max
    {
        result=result+R(q,t,k)(1 − p)^q
    }
}
E(Y)=result
```

a 3-dimensional table. We fill this table according to increasing values of ℓ. This works because computing $R(q,t,\ell)$ requires only already computed values of the form $R(q',t',\ell-1)$. For a given ℓ, we must iterate through the possible values of t from 0 to n, and for each such t, we must fill the table for all possible values of q from q_{min} to q_{max}. See Algorithm 1 for details.

As a starting point, if $\ell = 1$, then for all values of t, T_t consists of a single partial coloring in which all vertices are assigned the same single color. Therefore, if $\ell = 1$, then $q_{min} = q_{max} = \binom{t}{2}$ and for this value of q we have $R(q,t,1) = 1$. As additional boundary conditions, we have $R(q,t,\ell) = 0$ in all cases when $t < 0$ or $q < q_{min}$.

Since $t = O(n)$, $j = O(n)$, $q_{max} = O(n^2)$, and $\ell = O(k)$, the runtime of Algorithm 1 is $O(kn^4)$. This is polynomial in the size of the graph, though quite high. On the other hand, the calculation of the $R(q,t,\ell)$ values is the most time-consuming part of the algorithm, and these values can be pre-computed and stored. Afterwards, we can compute $E(Y)$ more quickly – namely in $O(n^3)$ steps – for different values of n, p, k.

5. THE EXPECTED VALUE OF THE NUMBER OF SOLUTIONS

As a by-product of the presented model for algorithm performance, we also obtain results on the expected number of solutions. This is because the number of solutions is exactly $S = \sum_{w \in T_n} Y_w$, and thus the expected number of solutions is $E(S) = \sum_{w \in T_n} (1-p)^{q(w)}$. As previously, this sum has exponentially many terms, but again, the terms can be grouped according to the value q among the $q(w)$ values. With the notation introduced previously, we can write it as

$$E(S) = \sum_{q=q_{min}}^{q_{max}} R(q,n,k)(1-p)^q.$$

Thus we can compute $E(S)$ with a slight modification of Algorithm 1 in $O(kn^4)$ time. If the $R(q,t,k)$ values are already pre-computed, then computing $E(S)$ takes only $O(n^2)$ time, since the above formula for $E(S)$ has less than q_{max} terms and $q_{max} = O(n^2)$.

Recalling that our runtime prediction is accurate for non-k-colorable graphs only, the expected number of solutions can help us to decide, in what range of the parameters our runtime prediction is accurate. If the expected number of solutions is very small, then probably there is no solution and hence our runtime estimation is accurate, whereas if the expected number of solutions is high, then probably the graph is k-colorable, and our runtime prediction is only an upper bound on the real value. More precisely, knowing the expected number of solutions allows us to estimate the probability that a problem instance is solvable using Markov's inequality:

$$Pr(\text{solvable}) = Pr(S \geq 1) \leq E(S).$$

What is more, the probability of solvability can also be bounded from below using the first and second moments of S [1]. Practically, if the problem instance parameters are such that $E(S)$ is significantly less than 1, then such problem instances are probably unsolvable. If, on the other hand, $E(S)$ is significantly above 1, then the problem instances are probably solvable. That is, the phase transition will be near the point where $E(S) \approx 1$.

6. NUMERICAL RESULTS

This section shows some simulation results based on the presented method.

6.1 Size of the search tree

The method presented in Section 4 enables us to gain some insight as to how the complexity of graph coloring changes for different values of the parameters n, k, and p. Figure 3 shows an example: $E(Y)$ as a function of n and k, for fixed p. We can conclude from the figure that for small values of k, the problem is easy, even if n becomes large. This is consistent with previous results on the relatively low average-case complexity of graph coloring [27, 26]: although the complexity is exponential in n in the worst case, but it is $O(1)$

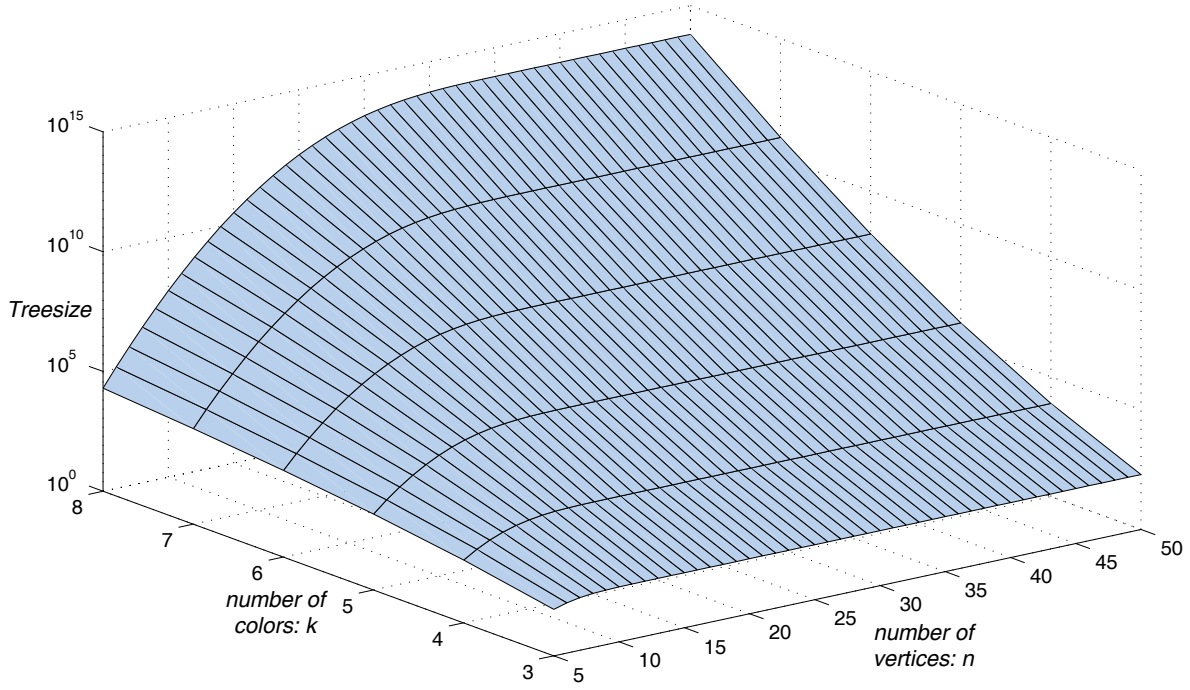

Figure 3: Expected size of the search tree for $p = 0.5$, as a function of n and k.

in the average case. However, as k increases, this increases the complexity of the problem dramatically (note the exponential scale on the vertical axis). It is still true that the complexity saturates, i.e. increasing n does not increase the complexity significantly after some threshold. However, this saturation takes place at a much higher value than in the case of small k.

The same phenomenon is depicted in Figure 4 from a different perspective. Here, n is fix, and p and k are varied. Again, it can be seen from the figure, that the complexity is in many cases quite low and hardly increasing with growing k. However, there is again a range of the parameters in which the complexity explodes. This is in line with the practical experience of high variability in the runtime of the algorithm. It is also clear that the curve must be monotonously decreasing in p: this is because in the non-k-colorable case, where our algorithm model is accurate, increasing p makes it easier for the algorithm to prove uncolorability, as more edges are likely to make the contradiction apparent earlier on (at a higher level of the search tree).

6.2 Number of solutions

Using the method presented in Section 5, we can also look at the expected number of solutions, and thus the picture can be further refined. Figure 5 depicts the expected number of solutions together with the expected size of the search tree for fix n and k, as a function of p. Since the complexity is exactly the number of all valid partial colorings in the search tree, and the number of solutions is the number of valid colorings on the nth level of the search tree, the figure shows clearly the changing contribution of the nth level of the search tree to the total search tree size. As can be seen, for small values of p, the search tree is dominated by the nth level.

However, as p increases, the contribution of the nth level decreases rapidly. This is again a consequence of the fact that the increased number of edges let the algorithm detect inconsistencies earlier on, thus it becomes rare that the algorithm actually reaches the nth level.

As mentioned earlier, our results are only accurate for non-k-colorable graphs, and the transition between k-colorability and non-k-colorability occurs roughly where the expected number of solutions is 1. Figure 5 also shows where this happens: for the used parameters, it is around $p \approx 0.5$. Hence, for the given values of n and k, our results are accurate in the $p > 0.5$ region. In other words, the curve in Figure 5 that shows the expected size of the search tree is accurate only in the right half of the diagram; in the left half, is is only an upper bound on the algorithm's runtime complextiy.

Finally, Figure 6 depicts the expected number of solutions together with the expected size of the search tree for fix p and k, as a function of n. As can be seen, for small values of n (in the range of $2k \ldots 3k$), the expected number of solutions is relatively high, i.e. the contribution of the nth

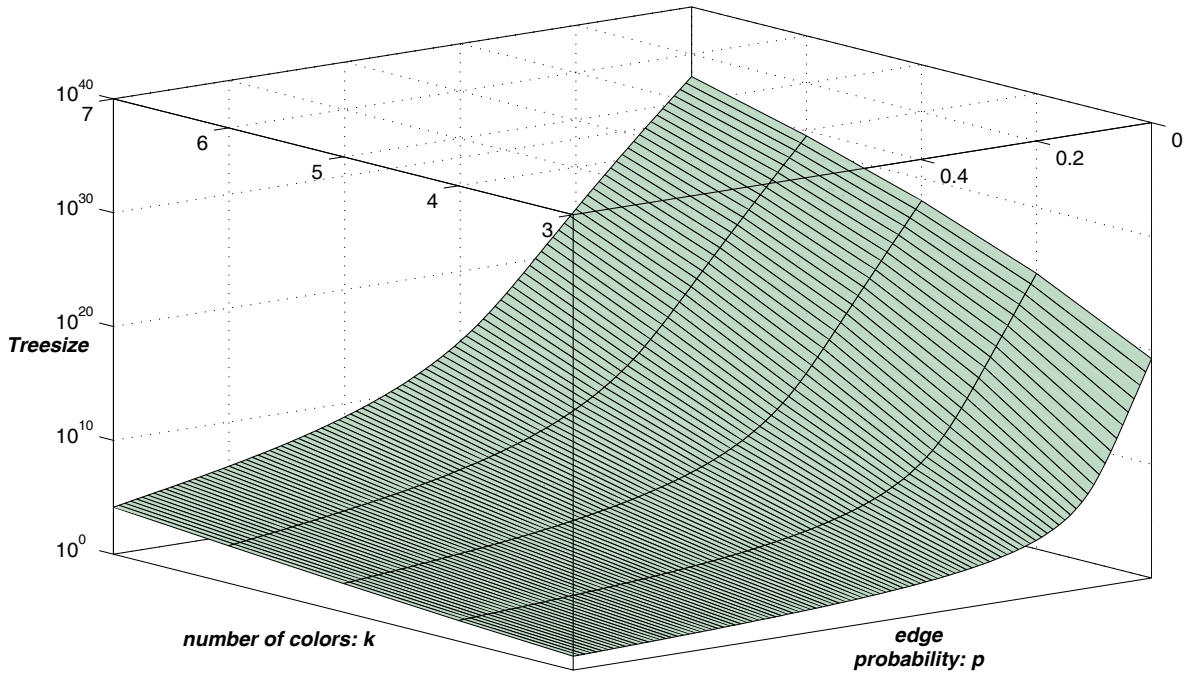

Figure 4: Expected size of the search tree for $n = 40$, as a function of p and k.

level of the search tree is high. In this region, increasing n significantly increases the algorithm's runtime. This is logical, because if n increases, then the algorithm will also visit many nodes on level $n+1$ of the search tree. However, after a while, the already mentioned saturation takes place. The expected number of solutions becomes very small, indicating that the algorithm rarely gets to the nth level of the search tree, as it usually finds a contradiction much earlier, without descending that far in the search tree. Accordingly, further increasing n does not significantly increase the complexity anymore, because the algorithm visits the lower parts of the search tree rarely anyway. This phenomenon reveals an interesting and quite complex connection between the algorithm's runtime complexity and the number of solutions.

7. CONCLUSION AND FUTURE WORK

In this paper, we have investigated the runtime complexity of a typical backtracking algorithm for coloring random graphs of the class $G_{n,p}$ with k colors. Using the expected size of the search tree as the measure of complexity, we devised a polynomial-time algorithm for predicting the backtrack algorithm's runtime complexity. As a by-product, our method also delivers the expected number of solutions of the given problem instance, which is interesting in its own right, but also helps to quantify when our model of runtime complexity is accurate.

Using the developed methods, we analyzed numerically how the algorithm's runtime complexity depends on the input parameters n, p, and k. We obtained a rich picture with regions of very low and very high complexity, and varying sensitivity with respect to changes in the input parameters. This way, our model can explain several of the phenomena that had been discovered before about the behaviour of backtrack-style optimization algorithms on graph coloring and related problems.

We also showed the multifaceted connection between the expected complexity of the problem and the expected number of solutions.

The most important limitation of the approach presented in this paper is that it is only accurate for non-k-colorable problem instances. Our future work will focus on extending the presented results to k-colorable problem instances. The main challenge of this is to take into account the order in which the algorithm visits the children of a node of the search tree, because this can have significant impact on the algorithm's running time. This difference might make it necessary to use different and/or more sophisticated methods to derive similar results for the case of k-colorable graphs as well.

Acknowledgements

This work was partially supported by the Hungarian National Research Fund and the National Office for Research and Technology (Grant Nr. OTKA 67651).

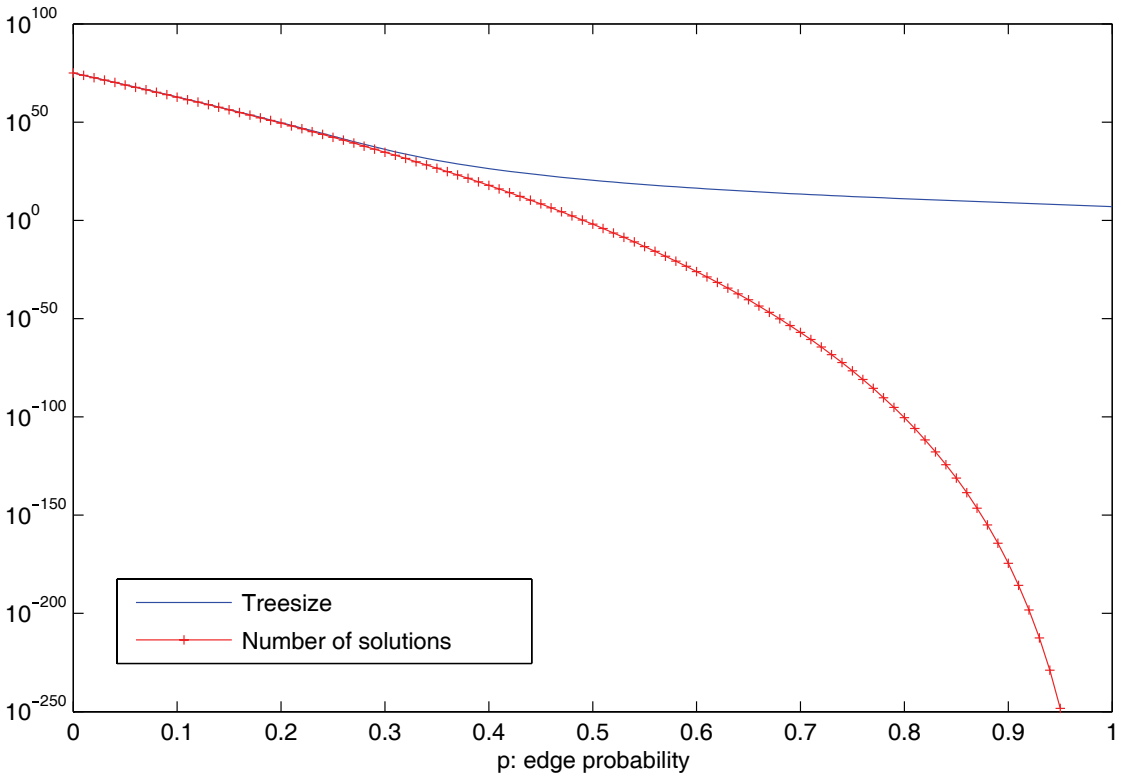

Figure 5: Expected number of solutions and expected search tree size for $n = 75$ and $k = 10$, as a function of p.

8. REFERENCES

[1] D. Achlioptas and A. Naor. The two possible values of the chromatic number of a random graph. In *36th ACM Symposium on Theory of Computing (STOC '04)*, pages 587–593, 2004.

[2] N. Alon and M. Krivelevich. The concentration of the chromatic number of random graphs. *Combinatorica*, 17(3):303–313, 1997.

[3] E. A. Bender and H. S. Wilf. A theoretical analysis of backtracking in the graph coloring problem. *Journal of Algorithms*, 6(2):275–282, 1985.

[4] B. Bollobás. The chromatic number of random graphs. *Combinatorica*, 8(1):49–55, 1988.

[5] D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

[6] P. Briggs, K. D. Cooper, and L. Torczon. Improvements to graph coloring register allocation. *ACM Transactions on Programming Languages and Systems*, 16(3):428–455, 1994.

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

[8] J. Culberson and I. Gent. Frozen development in graph coloring. *Theoretical Computer Science*, 265(1-2):227–264, 2001.

[9] P. Erdős and A. Rényi. On the evolution of random graphs. *Magyar Tud. Akad. Mat. Kutató Int. Közl*, 5:17–61, 1960.

[10] M. R. Garey and D. S. Johnson. The complexity of near-optimal graph coloring. *Journal of the ACM*, 23:43–49, 1976.

[11] M. R. Garey, D. S. Johnson, and L. J. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[12] C. P. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1-2):43–62, 2001.

[13] G. R. Grimmett and C. J. H. McDiarmid. On colouring random graphs. *Mathematical Proceedings of the Cambridge Philosophical Society*, 77(2):313–324, 1975.

[14] T. Hogg. Refining the phase transition in combinatorial search. *Artificial Intelligence*, 81(1-2):127 – 154, 1996.

[15] T. Hogg and C. P. Williams. The hardest constraint problems: A double phase transition. *Artificial Intelligence*, 69(1-2):359–377, 1994.

[16] H. Jia and C. Moore. How much backtracking does it take to color random graphs? rigorous results on heavy tails. In *Principles and Practice of Constraint Programming (CP 2004)*, pages 742–746, 2004.

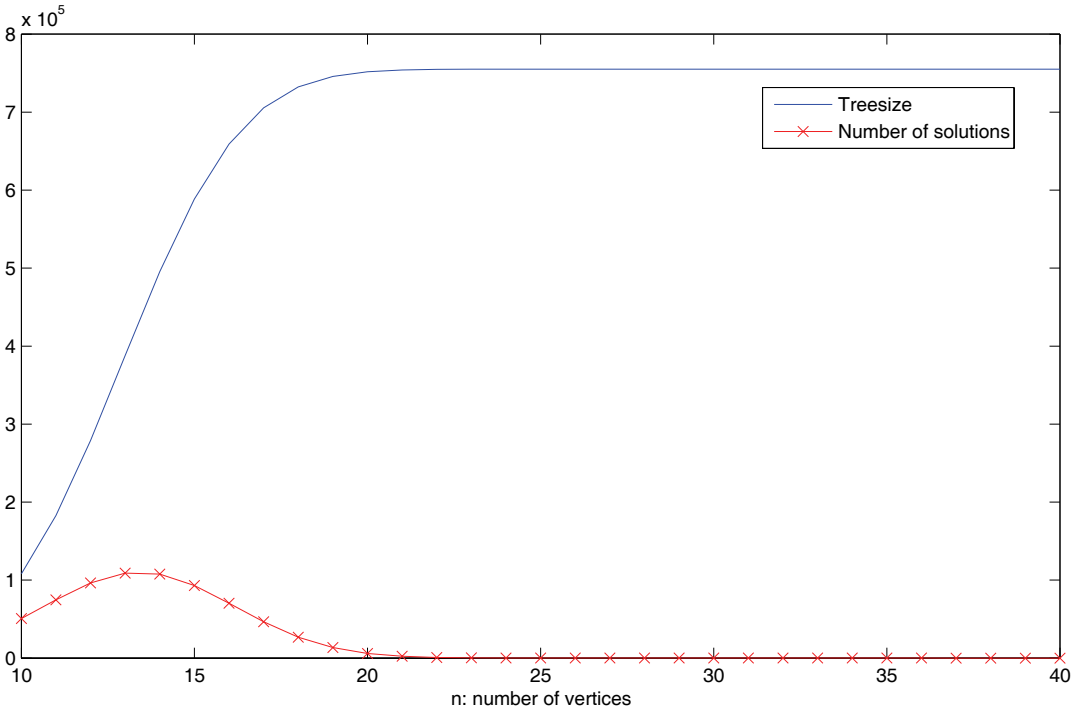

Figure 6: Expected number of solutions and expected search tree size for $p = 0.5$ and $k = 5$, as a function of n.

[17] T. Luczak. The chromatic number of random graphs. *Combinatorica*, 11(1):45–54, 1991.

[18] T. Luczak. A note on the sharp concentration of the chromatic number of random graphs. *Combinatorica*, 11(3):295–297, 1991.

[19] Z. Mann and A. Orbán. Optimization problems in system-level synthesis. In *3rd Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications*, pages 222–231, 2003.

[20] Z. Mann and A. Szajkó. Improved bounds on the complexity of graph coloring. In *12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, 2010.

[21] Z. Mann and T. Szép. BCAT: A framework for analyzing the complexity of algorithms. In *8th IEEE International Symposium on Intelligent Systems and Informatics*, pages 297–302, 2010.

[22] N. K. Mehta. The application of a graph coloring method to an examination scheduling problem. *Interfaces*, 11(5):57–65, 1981.

[23] R. Monasson. On the analysis of backtrack procedures for the coloring of random graphs. In E. Ben-Naim, H. Frauenfelder, and Z. Toroczkai, editors, *Complex Networks*, pages 235–254. Springer, 2004.

[24] E. Shamir and J. Spencer. Sharp concentration of the chromatic number on random graphs $G_{n,p}$. *Combinatorica*, 7(1):121–129, 1987.

[25] T. Szép and Z. Mann. Graph coloring: the more colors, the better? In *11th IEEE International Symposium on Computational Intelligence and Informatics*, 2010.

[26] J. S. Turner. Almost all k-colorable graphs are easy to color. *Journal of Algorithms*, 9(1):63–82, 1988.

[27] H. S. Wilf. Backtrack: an O(1) expected time algorithm for the graph coloring problem. *Information Processing Letters*, 18:119–121, 1984.