Determining the expected runtime of exact graph coloring*

Zoltán Ádám Mann and Anikó Szajkó
Budapest University of Technology and Economics
Department of Computer Science and Information Theory
Magyar tudósok körútja 2., 1117 Budapest, Hungary
e-mail: zoltan.mann@cs.bme.hu, 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 graph's size, density, and the number of available colors.

1 Introduction and previous work

Graph coloring 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 [16, 5, 15]. Unfortunately, graph coloring is NP-complete [8]. Moreover, if $P \neq NP$, then no polynomial-time approximation algorithm with an approximation factor smaller than 2 can exist for graph coloring [7].

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. Although in the worst case the backtrack algorithm requires an exponential number of steps, its average-case complexity is $O(1)$ [19].

The probabilistic analysis of the coloring of random graphs was first suggested in the seminal paper of Erdős and Rényi [6]. Through subsequent work of several researchers, the coloring and, in particular, the chromatic number of random graphs is well understood [10, 4, 11, 17, 12, 2, 1]. In terms of the performance of backtracking on random graphs, only some lower and upper bounds are known on the moments of the distribution of the algorithm's runtime [3]. However, as the difference between the known lower and upper bounds is quite high, it is not possible to predict even the order of magnitude of the runtime of backtracking on a problem instance.

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 [9]. It also enhances load balancing when several problem instances are solved in parallel on multiple machines.

Hence, our aim is to obtain accurate results on the expected runtime of the backtrack algorithm in coloring random graphs. We restrict ourselves to the non-colorable case; extension of our model to the colorable case remains as future work. We use the size of the search tree as a measure of complexity and analyze the expected size of the search tree as a function of input parameters. Our contribution is an algorithm for determining the expected size of the search tree exactly. The algorithm uses dynamic programming, and its runtime is polynomial in the size of the graph. We also present our empirical findings on how the complexity of the problem depends on the input parameters.

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

*This paper was presented in: *Mini-conference on Applied Theoretical Computer Science (MATCOS), Koper (Slovenia), 2010.* It was published in: *Proceedings of the 13th International Multiconference „Information Society – IS 2010", Volume A, pages 389-393, 2010.*

$1, \ldots, k$. 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 a color. This is the root – that is, the single node on level 0 – of the search tree. 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 levels, the last level containing the colorings of the graph. Let T_t denote the set of partial colorings on level 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.

In each partial coloring w, the backtrack algorithm considers the children of w and visits only those that are valid. 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 given problem instance.

3 The expected number of visited nodes of T

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:

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

Let $Q(w) := \{\{x, y\} \in V^2 : x \neq y, color(x) = color(y)\}$, where V^2 is the set of unordered pairs of elements of V. 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 $E(Y)$ through this formula is not tractable since $|T|$ is exponentially large in n.

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.

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

□

It is also possible to derive a formula for the minimum of $q(w)$ [13], but it 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 nodes in T_t.

If we could determine all the $R(q, t, k)$ values explicitly, that would enable us to calculate the exact value of $E(Y)$:

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

Determining the $R(q, t, k)$ values is possible with the following recursion:

Proposition 2.

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

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 $k - 1$ 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 $q - \binom{j}{2}$. □

Based on this recursive formula, we can use dynamic programming to compute the $R(q, t, k)$ values and store them in a 3-dimensional table. We fill this table according to increasing values of k. For a given k, we must iterate through

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

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

for k=2 to number of colors
{
    for t=0 to n
    {
        for q=q_min to q_max
        {
            R(q,t,k) = 0
            for j=0 to t
            {
                if q - C(j,2) ≥ q_min(t-j, k-1)
                {
                    R(q,t,k) = R(q,t,k) + C(t,j) R(q - C(j,2), t-j, k-1)
                }
            }
        }
    }
}

k=number of colors
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
```

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}. As a starting point, when $k = 1$, then for all values of t, $q_{min} = q_{max} = \binom{t}{2}$ and for this value of q we have $R(q, t, k) = 1$. As additional boundary conditions, we have $R(q, t, k) = 0$ in all cases when $t < 0$ or $q < q_{min}$. See Algorithm 1 for details.

Since $t = O(n)$, $j = O(n)$ and $q_{max} = O(n^2)$, 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 $R(q, t, k)$ 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 for different values of n, p, k.

5 Numerical results

The presented method enables us to gain some insight as to how the complexity of graph coloring changes for different values of the parameters n, k, p. Fig. 1 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 [19, 18]. 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.

A more detailed empirical analysis using the tool BCAT [14] will be part of a future extended version of this paper.

6 Conclusion and future work

We have investigated the 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 complexity.

In this paper, we only dealt with uncolorable problem instances. Our future work will focus on extending the presented results to colorable problem instances.

Acknowledgements

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

References

[1] Dimitris Achlioptas and Assaf 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] Noga Alon and Michael Krivelevich. The concentration of the chromatic number of random graphs. *Combinatorica*, 17(3):303–313, 1997.

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

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

[5] Preston Briggs, Keith D. Cooper, and Linda Torczon. Improvements to graph coloring register allocation. *ACM Transactions on Programming Languages and Systems*, 16(3):428–455, 1994.

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

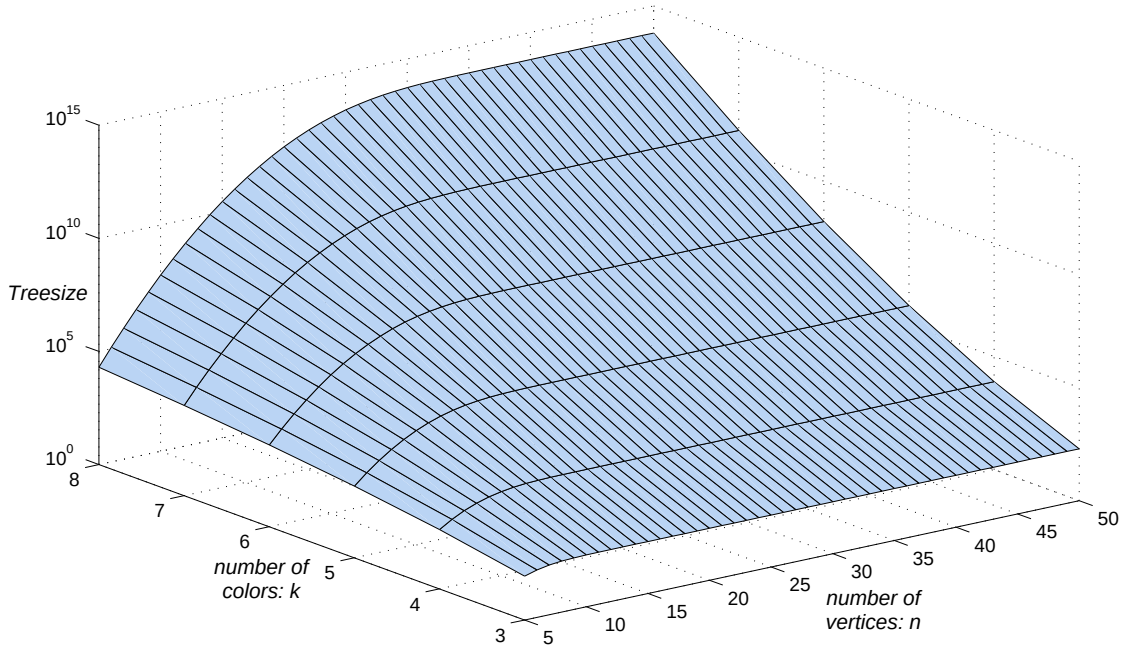

Figure 1: Expected size of the search tree for $p = 0.5$, as a function of n and k.

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

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

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

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

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

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

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

[14] Zoltán Á. Mann and Tamás Szép. BCAT: A framework for analyzing the complexity of algorithms. In *8th IEEE International Symposium on Intelligent Systems and Informatics*, 2010.

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

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

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

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

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