Sublinear colorings of 3-colorable graphs in linear time

Thomas Tseng
Carnegie Mellon University
Pittsburgh, Pennsylvania
tomtseng@cmu.edu

ABSTRACT

There has been extensive research on developing algorithms for finding good colorings of 3-colorable graphs in polynomial time. In this paper, we impose an even stricter running time requirement: our algorithm must find colorings in linear time with respect to the number of vertices. This means that if the graph is dense, we cannot even afford to look at all of the edges. We show that in the word RAM model, we can color a 3-colorable graph with $O(n/\log\log n)$ colors in $O(n)$ work and $O(\log\log n)$ span.

CCS CONCEPTS

• **Theory of computation → Graph algorithms analysis**; *Approximation algorithms analysis*; Parallel algorithms;

KEYWORDS

approximation algorithms, graph coloring

ACM Reference Format:
Thomas Tseng. 2018. Sublinear colorings of 3-colorable graphs in linear time. In *Proceedings of Special Interest Group on Harry Quimby Bovik (SIGBOVIK'18).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

The problem of determining whether a graph is 3-colorable is a well-studied NP-complete problem [1]. Many researchers have worked on polynomial-time algorithms for coloring 3-colorable graphs using as few colors as possible, with the most recent development being an algorithm that achieves $O\left(n^{.19996}\right)$ colors through a combinatorial approach combined with semidefinite programming [2].

An interesting extension that has use in neither theory nor practice is to stipulate a stronger running time requirement. In particular, we wonder what the best coloring achievable is using $O(n)$ running time. This means that we cannot even afford to look at most of the edges of a dense graph. Is it still possible to find a coloring with $o(n)$ colors?

We answer in the affirmative by giving an algorithm under the word RAM model that produces $O(n/\log\log n)$-colorings of 3-colorable graphs in $O(n)$ work. Moreover, our algorithm is massively parallel with $O(\log\log n)$ span.

SIGBOVIK'18, March 2018, Pittsburgh, Pennsylvania USA

ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

2 APPLICATIONS

3 PRELIMINARIES

Under the word RAM model, the machine on which our algorithm runs stores integers in *words*. The word size $w \geq \log_2 n$ scales with the problem size n, which for our purposes is the number of vertices in the input graph. This model allows us to perform bitwise and arithmetic operations on words in constant time.

To more closely follow the notation used in many programming languages for bitwise logical operators, we use & to denote bitwise conjunction, | to denote bitwise disjunction, and ~ to denote bitwise negation. Specifically, if we have two boolean vectors v and u of length ℓ, then the results of v & u, $v \mid u$, and $\sim v$ are all boolean vectors of length ℓ such that

$$(v \mathbin{\&} u)_i = v_i \wedge u_i, \qquad (v \mid u)_i = v_i \vee u_i, \qquad (\sim v)_i = \neg v_i.$$

When A is a matrix and v is a vector, $A \cdot v$ represents boolean matrix multiplication, that is,

$$(A \cdot v)_i = \bigvee_j A_{i,j} \wedge v_j.$$

4 ALGORITHM

Let the input graph be given in adjacency matrix format. We assume the input graph is 3-colorable, which implies that any subgraph of the graph is also 3-colorable. Given a parameter k, consider partitioning the vertices into n/k contiguous chunks of k vertices. If we can 3-color the subgraph induced by each of the n/k chunks in $O(k)$ time, we can combine all these 3-colorings to achieve a $3n/k \in O(n/k)$-coloring for the whole graph in $O(n)$ time. We pick $k = \log_4 w \in \Omega(\log\log n)$, so $3^k(k+1) \leq w$ for sufficiently large w (and hence for sufficiently large n). With this setting of k, we indeed can 3-color each subgraph in $O(k)$ time with the help of word-level parallelism.

Algorithm 1 Sublinear coloring algorithm

```
1: procedure Color(M)
2:     Do everything described in the text below
3:     return the resulting coloring
4: end procedure
```

We can represent a 3-coloring of a graph of k vertices by three k-length bit vectors. The j-th bit of the i-th vector is set if and only if the j-th vertex has color i. The idea here is that if we have the three k-length bit vectors $v^{(0)}, v^{(1)}, v^{(2)}$ representing a 3-coloring as well as the adjacency matrix A of a k-vertex graph, we can check that the coloring is valid for the graph by checking that $\left(A \cdot v^{(i)}\right) \mathbin{\&} v^{(i)} = \vec{0}$ for each i. This is because the j-th bit of $A \cdot v^{(i)}$ is set if the j-th vertex has any neighbors of color i, so then ANDing with $v^{(i)}$ tells

us about which i-colored vertices have i-colored neighbors. Due to how small k is, we can check all 3-colorings for validity in parallel.

We start by precomputing some constants to be reused for all of the subgraphs. Because $3^k(k+1) \leq w$ for sufficiently large n, we can pack the aforementioned representation of all 3^k possible 3-colorings into three words $u^{(0)}, u^{(1)}, u^{(2)}$ with a bit of room to spare for each coloring. Each word $u^{(i)}$ is broken into 3^k blocks where each block is $(k+1)$ bits wide. The k-length bit vector for the i-th color of the j-th possible 3-coloring is the low order bits of the j-th block of $u^{(i)}$. We also precompute B_{H} to be a word broken into the same 3^k blocks where each block has only its high-order bit set, and precompute B_{L} to be a word broken in 3^k blocks where each block has only its low-order bit set.

Iterate over each chunk of k vertices and do the following: consider the subgraph induced by the k vertices. We proceed to perform the parallel boolean matrix multiplication. For each $r = 0, 1, \ldots, k-1$, we fetch the r-th row of the $k \times k$ adjacency matrix in constant time by jumping to the appropriate place in the input and doing some shifting and bit masking. Multiply the word by B_{L} so that we now have a word w_r consisting of 3^k copies of row r of the adjacency matrix. Now $w_r \ \& \ u^{(i)}$ is a word of 3^k blocks where the j-th block is non-zero if and only if the r-th entry of the corresponding boolean matrix product is non-zero. Then $z_{r,i} = \left(\sim \left(B_{\mathrm{H}} - \left(w_r \ \& \ u^{(i)}\right)\right)\right) \ \& \ B_{\mathrm{H}}$ is a word of 3^k blocks where the j-th block has its high-order bit set if and only if the r-th entry of the corresponding boolean matrix product is non-zero. Computing each $z_{r,i}$ is constant time, so computing all of them takes $O(k)$ time. Shift and OR the $z_{r,i}$'s together appropriately to get words $y^{(i)}$ of 3^k blocks where the j-th block has the result of the boolean matrix product corresponding to color i of the j-th coloring. Compute $y = \left(y^{(0)} \ \& \ u^{(0)}\right) \mid \left(y^{(1)} \ \& \ u^{(1)}\right) \mid \left(y^{(2)} \ \& \ u^{(2)}\right)$, which has that the j-th block is all zeroes if the j-th coloring is valid. Compute $x = (B_{\mathrm{H}} - y) \ \& \ B_{\mathrm{H}}$, which has that its j-th block has its high-order bit set to 1 if the j-th coloring is valid. Binary search for a set bit in x in $O(\log w) = O(k)$ time using lots of masking, and after finding that bit, we read off a 3-coloring for the subgraph by indexing appropriately into $u^{(0)}, u^{(1)}, u^{(2)}$. This is all $O(k)$ time for a chunk of k vertices.

We do this for n/k chunks of k vertices, so this takes $n/k \cdot O(k) = O(n)$ time. By using a different set of three colors for each subgraph, the number of colors used over the whole graph is $3n/k \in O(n/\log\log n)$ as desired. We also see that we achieve $O(k) = O(\log\log n)$ span if we use some parallelism in precomputing $u^{(0)}, u^{(1)}, u^{(2)}, B_{\mathrm{L}}, B_{\mathrm{H}}$ and if we iterate over all n/k chunks of vertices in parallel.

5 EXPERIMENTS

We implement our algorithm in C++ and measure its speedup. Unlike in the idealized word RAM model, we do not have machines that scale their word size to input sizes. Instead, our code uses a fixed word size of 32 bits. With this, we output $3n/4$-colorings of 3-colorable graphs.

We run our implementation on a 40-core machine with $4 \times$ 2.4GHz Intel 10-core E7-8870 Xeon processors and 256GB of main memory. We compile our code with g++ version 5.3.0 and use Cilk Plus extensions [3] to support parallelism. A version of our code that uses OpenMP for parallelism instead of Cilk Plus is available at https://github.com/tomtseng/sublinear-coloring.

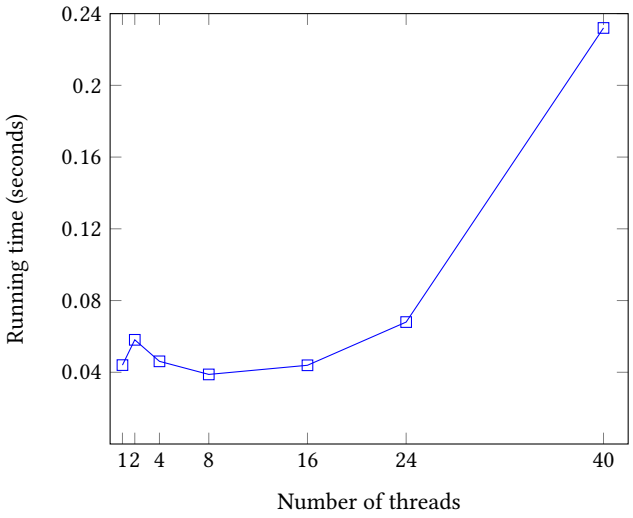

Figure 1: Running time of our implementation

As our input graph for our experiments, we use the most 3-colorable of all graphs: a graph of 200,000 vertices with no edges. We cannot use graphs with many more vertices due to how memory-intensive it is to allocate and store adjacency matrices.

Our running time using various numbers of threads is plotted in figure 1. The speedup curve looks good, except that it goes in the wrong direction.

6 FUTURE WORK

Some open questions to explore in the area of coloring 3-colorable graphs in $O(n)$ time include the following:

- Our algorithm crucially relies on the word RAM model by using word-level parallelism to obtain time savings. Can we achieve $o(n)$-colorings in other computational models?
- Is it possible to achieve a truly sublinear coloring, that is, an $O\left(n^{1-\varepsilon}\right)$-coloring for some constant $\varepsilon > 0$?
- What lower bounds can we prove assuming this O(n) running time restriction?

7 ACKNOWLEDGMENTS

This problem of achieving the best graph coloring possible in $O(n)$ time was proposed by some of the Spring 2017 15-251 teaching assistants at Carnegie Mellon University during a particularly unproductive grading session.

REFERENCES

[1] Michael R Garey, David S. Johnson, and Larry Stockmeyer. 1976. Some simplified NP-complete graph problems. *Theoretical computer science* 1, 3 (1976), 237–267.
[2] Ken-ichi Kawarabayashi and Mikkel Thorup. 2014. Coloring 3-colorable graphs with $o\left(n^{1/5}\right)$ colors. In *LIPIcs-Leibniz International Proceedings in Informatics*, Vol. 25. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
[3] Charles E Leiserson. 2009. The Cilk++ concurrency platform. In *Proceedings of the 46th Annual Design Automation Conference*. ACM, 522–527.