$o(n)$-coloring 3-colorable graphs in $O(n)$ time

Tom Tseng

In this writeup, we give an algorithm that takes as input a 3-colorable graph in adjacency matrix format and outputs an $O(n/\log\log n)$ coloring in $O(n)$ time under the word RAM model.

1 Notation

We refer to the number of vertices of the input graph as n. We'll let $w \geq \log_2 n$ denote the word size. We'll assume $w \in \Theta(\log n)$, because if we had $w \in \omega(\log n)$, we could just ignore the higher bits of every word.

To more closely follow the notation used in many programming languages for bitwise logical operators, we'll use & to denote bitwise conjunction (AND), | to denote bitwise disjunction (OR), and $\sim$ to denote bitwise negation (NOT). More specifically, if we have two boolean vectors v and w of length ℓ, then the results of $v \& w$, $v \mid w$, and $\sim v$ are all boolean vectors of length ℓ such that

$$(v \;\&\; w)_i = v_i \wedge w_i, \qquad (v \mid w)_i = v_i \vee w_i, \qquad (\sim v)_i = \neg v_i.$$

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

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

We'll use "1" and "true" interchangably and "0" and "false" interchangably.

2 Sketch of the algorithm

The high-level idea is as follows: we break the vertices into chunks of size k, and for the subgraph induced by each chunk, we 3-color the subgraph by brute force (using a different set of 3 colors for each subgraph). This gives a $3n/k$ coloring of the graph. We set k to be just a little bigger than a constant. To keep the runtime linear, we use some bit hackery to get some word-level parallelism.

Let the input be given in adjacency matrix format. Pick $k \in \Omega(\log\log n)$ such that $3^k(k+1) \leq w$, e.g. $k = \log_4 w$ will suffice. We partition the vertices into n/k contiguous chunks of k, and if we can 3-color the subgraph induced by each chunk in $O(k)$ time, then we'll achieve our desired result.

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 vector as 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 the k-vertex graph, we can check that the coloring is valid by checking that $(A \cdot v_i) \;\&\; v_i = \vec{0}$ 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 possible 3-colorings for validity in parallel.

We start by precomputing some constants. Because k is so small, we can pack a the aforementioned representation of all 3^k colorings into three words u_0, u_1, u_2. 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 right-aligned within the j-th block of u_i. We also precompute B_{L} to be a word broken into the same 3^k blocks where each block holds just a single left-aligned 1 bit, and precompute B_{R} to be a word broken in 3^k blocks where each block holds just a single right-aligned 1 bit.

Iterate over each chunk of k vertices and perform the following: consider the subgraph induced by the k vertices. We'll perform the parallel boolean matrix multiplication now. 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 bit masking and shifting. Multiply the word by B_{R} 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 iff the r-th entry of the corresponding boolean matrix product is non-zero. Then $z_{r,i} = (\sim(B_{\mathrm{L}} - (w_r \,\&\, u_i))) \,\&\, B_{\mathrm{L}}$ is a word of 3^k blocks where the j-th block has its high-order bit set iff 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 each $z_{r,i}$ 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 $y = (y_0 \,\&\, u_0) \mid (y_1 \,\&\, u_1) \mid (y_2 \,\&\, u_2)$, which has that the j-th block is all zeroes if the j-th coloring is valid. Compute $x = (B_{\mathrm{L}} - y) \,\&\, B_{\mathrm{L}}$, 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 $O(\log w) = O(k)$ using lots of masking, and after finding that bit, we can 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. The number of colors used over the whole graph is $3n/k = O(n/\log\log n) \subseteq o(n)$.