Parallel and On-line Graph Coloring

Magnús M. Halldórsson
Science Institute
University of Iceland
IS-107 Reykjavik, Iceland*

Abstract

We discover a surprising connection between graph coloring in two orthogonal paradigms: parallel and on-line computing. We present a randomized on-line coloring algorithm with a performance ratio of $O(n/\log n)$, an improvement of $\sqrt{\log n}$ factor over the previous best known algorithm of Vishwanathan. Also, from the same principles, we construct a parallel coloring algorithm with the same performance ratio, for the first such result. As a byproduct, we obtain a parallel approximation for the independent set problem.

1 Introduction.

At the heart of most partition and allocation problems is a graph coloring question. The graph coloring problem involves assigning values (or colors) to the vertices of a graph so that adjacent vertices are assigned distinct colors, with the objective of minimizing the number of colors used.

We seek graph coloring algorithms that are both *effective* and *efficient*. Efficiency means, among other things, using computation time bounded by a polynomial in the size of the input. Effectiveness means that the quality of the coloring is never much worse than the optimal coloring; it is measured in terms of the *performance ratio* of the algorithm, defined as the ratio of the number of colors used to the minimum number of colors needed, maximized over all inputs. In this paper we present a reasonably effective and efficient graph coloring algorithm that yields both parallel and on-line algorithms.

A *parallel* algorithm is said to be efficient if it uses time polylogarithmic and processors polynomial in the size of the input. The coloring problem is of central importance in parallel and distributed computation, as a means of scheduling non-conflicting concurrent tasks. Previous parallel algorithm have attained the degree bound of Brooks's theorem [9], 5-coloring of planar graphs [6], and some other special cases, but no non-trivial performance guarantee on general graph was known. We present a parallel algorithm that uses at most $3n^{(k-2)/(k-1)}$ colors on k-colorable graphs, for a performance ratio of $O(n/\log n)$, where n is the number of vertices.

An algorithm is said to be *on-line* if the color assigned to vertex i is independent of vertices with index greater than i. This corresponds to various dynamic situations, such as the assignments of frequencies to mobile users. The on-line graph coloring problem has been widely studied [1, 8, 14] and for various other reasons as well. In particular, an algorithm that improves the trivial performance ratio to $O(n/\log^* n)$ was given by Lovász, Saks, and Trotter [15], and a randomized algorithm with ratio of $O(\sqrt{n})$ on 3-colorable graph and $O(n/\sqrt{\log n})$ ratio on

*email: `mmh@rhi.hi.is`. Research partly performed at Rutgers University, Tokyo Institute of Technology, and Japan Advanced Institute of Science and Technology – Hokuriku, and supported there by a DIMACS Graduate Fellowship, a Toshiba Endowed Chair, and a PFU Endowed Chair, respectively.

general graphs was given by Vishwanathan [17]. We improve here the randomized algorithm of [17], for a performance ratio of $O(n/\log n)$.

We first present in Section 2 a sequential graph coloring algorithm with a performance ratio of $O(n/\log n)$, and use it to derive both a parallel algorithm (in Section 3) and a randomized on-line algorithm (in Section 4) with the same performance ratio. The randomized algorithm is an improvement of an algorithm of Vishwanathan [17], and the performance ratio of $O(n/\sqrt{\log n})$ compares favorably with the lower bound for randomized on-line coloring of $\Omega(n/\log^2 n)$ [11]. The performance ratio of the parallel algorithm is the first of its kind. It also compares favorably with the best performance ratio known for sequential coloring of $O(n(\log\log n)^2/\log^3 n)$ [10], and the best announced lower bound known for that quantity (assuming $P \neq NP$) of $n^{1-\epsilon}$, for each $\epsilon > 0$ [5]. Finally, using the parallel coloring algorithm, we obtain in Section 5 a $O(n(\log\log n/\log n)^2)$ performance ratio for parallel independent set problem. This also compares favorably to the sequential upper bound of $O(n/\log^2 n)$ [3] and the announced lower bound of $n^{1-\epsilon}$, for each $\epsilon > 0$ [12]. See Table 1 for comparisons.

	On-line		Parallel	Sequential
	Deterministic	Randomized		
Known upper bound	$O(n/\log^* n)$ [15]	$O(n/\sqrt{\log n})$ [17]		$O(n(\lg\lg n)^2/\log^3 n)$ [10]
Here		$O(n/\log n)$	$O(n/\log n)$	
Known lower bounds	$(\Omega(n/\log^2 n))$	$\Omega(n/\log^2 n)$ [11]	$(\Omega(n^{1-\epsilon}))$	$\Omega(n^{1-\epsilon})$ [5]

Table 1: New and known results on approximate graph coloring

This connection between apparently unrelated algorithmic paradigms is quite surprising, as the two paradigms we study are seemingly orthogonal and incompatible. Parallel computation is inherently off-line, and on-line computation inherently sequential. Nevertheless, there are certain strong similarities, in particular, both require a loose coupling among subproblems. Parallel algorithms benefit greatly from locality, which can be thought of as a case of limited information similar to the on-line regime. On-line algorithms know more about certain other subproblems, but with less control over what they learn.

2 Approximate Coloring Algorithms

In this section, we present and analyze the underlying sequential, off-line coloring algorithm that forms the basis of the parallel and on-line variants.

Let us first review what we have learned from previous methods. First, consider an algorithm due to Wigderson [19]. Given a three-colorable graph on n vertices, we look for a vertex of degree at least $\sqrt{n}$. The neighborhood of that vertex must be two-colorable, so we color these neighbors with two fresh new colors in linear time and remove them from the graph. We repeat this until maximum degree falls below $\sqrt{n}$, for at most $\sqrt{n}$ iterations. The remaining graph can easily be colored using (max degree) $+ 1 \leq \sqrt{n}$ colors, for a combined total of $3\sqrt{n}$ colors. This principle can be generalized to graphs of larger chromatic number, where the key observation is that the neighborhood of any node in a k-colorable graph must be $k-1$ colorable. This allows us the break the problem into a collection of somewhat easier subproblems.

The problem with this approach is that it fills the color classes in sequence. We need a method that accumulates many color classes concurrently, rather than searching for individual independent sets. We extract an idea from the on-line algorithm of Vishwanathan [17].

A simple greedy algorithm, sometimes referred to as *First Fit*, works as follows. The vertices are processed in sequence, according to some (arbitrary) order, and a vertex is assigned to the first color to which none of the preceding neighbors of the vertex have been assigned. If we are given a limited number of colors, it may not be possible to color all the vertices. The resulting coloring of a subset of the vertices is called a *maximal partial coloring* of the graph.

A maximal partial coloring is an assignment of some of the nodes into fixed number of "greedy" color classes so that each remaining node is adjacent to some node in each greedy class. The important point is that each color class partitions the remaining nodes into subsets of the neighborhoods of the nodes in the color class. If we choose a small class, the remaining uncolored nodes induce relatively few subproblems. The total number of colors used is then the number of greedy color classes, plus the sum of the number of colors used in the subproblems. Such a maximal partial coloring is easy to achieve sequentially, via the First Fit algorithm which assigns a vertex to the first compatible color class (if one exists).

```
K-Color($G$,$k$)
begin
  if ($k \le 2$) BipartiteColor($G$)
  else if ($k > \log n$) Assign each vertex a different color.
  else
    $s \leftarrow s(n,k) = 2^{1/(k-1)} (n/(k-2))^{(k-2)/(k-1)}$
    ResidueNodes $\leftarrow$ MaximalPartialColor($G$,$s$)
    Find the smallest greedy color class, and let $w_1, \ldots, w_p$ be its nodes.
    Partition the ResidueNodes into $R_1, \ldots R_p$ such that nodes in $R_i$ are adjacent to $w_i$.
    for $i = 1$ to $p$
      Color($R_i$, $k-1$)
end
```

Figure 1: An approximation algorithm for k-colorable graphs

We state the main algorithm in Figure 1. The algorithm finds a maximal partial coloring, partitions the remaining nodes around the smallest color class, and recurses on those subproblems, with the recursion bottoming out at bipartite graphs. When the chromatic number is too large for us to handle, with respect to the size of the graph, we settle on the trivial coloring of one color per vertex.

Theorem 2.1 *The number of colors used by* Color *on k-colorable graphs is at most $f(k) \cdot n^{(k-2)/(k-1)}$ where $f(k) = 1 + O(\frac{\log k}{k})$.*

Proof. Let $A(n,k)$ denote the maximum number of colors used by the algorithm to color any k-colorable graph on n vertices. Let $f(k)$ denote $2^{1/(k-1)}((k-1)/(k-2)^{(k-2)/(k-1)})$, and let $\mathcal{A}(n,k)$ denote $f(k)n^{(k-2)/(k-1)}$. Our proposition is that $A(n,k) \le \mathcal{A}(n,k)$, for all positive integers n, k, $k \ge 2$.

The proof proceeds by induction on k, with the base case, $k = 2$, established by an optimal bipartite algorithm. Hence, assume from here on that $A(n, k-1) \le \mathcal{A}(n, k-1)$, for all positive integers n.

The number of colors used by our algorithm is the number of greedy colors, $s = s(n,k)$, plus the number of colors used in the t subproblems, each on m_j vertices. (To maintain sanity and simplicity of presentation, we ignore all ceilings.) Note that $\mathcal{A}(n,k) = (k-1)s(n,k)$, so our main task is to show that the number of colors used in the subproblems be at most $(k-2)s(n,k) = (2(k-2)n)^{(k-2)/(k-1)}$. Intuitively, we find that the worst case occurs when the subproblems are all of equal size, and then derive the worst-case number of subproblems.

If g is the total number of vertices in the greedy color classes, then the number of vertices in the smallest class, and thus the number of subproblems, is at most g/s. By the inductive assumption, we can assume that the number of colors used in the subproblems is bounded by $\mathcal{A}$. Thus,

$$A(n,k) \leq s + \sum_{j=1}^{g/s} \mathcal{A}(m_j, k-1)$$

Since the approximation function is convex, the sum is maximized when the subproblem sizes are all equal.

$$A(n,k) \leq s + \max_g\{\frac{g}{s} \cdot \mathcal{A}(\frac{\sum_j m_j}{g/s}, k-1) \mid n = g + \sum_j m_j\}$$

Substitution and simplification gives

$$A(n,k) \leq s + \max_g\{g \cdot (\frac{n-g}{g})^{(k-3)/(k-2)}\} \cdot 2^{1/(k-2)} \frac{k-2}{(k-3)^{(k-3)/(k-2)}} \cdot s^{-1/(k-2)}$$

The function $f_k(g) = g \cdot (\frac{n-g}{g})^{(k-3)/(k-2)}$ attains a maximum value of $(k-3)^{(k-3)/(k-2)} n/(k-2)$, conveniently simplifying the formula. By straightforward algebraic manipulation, we find that

$$A(n,k) \leq s + 2^{1/(k-2)} n \cdot s^{-1/(k-2)} = \mathcal{A}(n,k).$$

∎

Our algorithm assumes that the chromatic number of the graph is given as input. To circumvent that, we can run the algorithm for all natural numbers less than n, and retain the best result. A more efficient binary search approach add only a $\log k = O(\log \log n)$ factor to the time complexity. Start with the range $[0,n]$, as the algorithm is guaranteed to fail for $k=0$ and succeed for $k=n$, and run the algorithm for the mid-value of the interval. If it succeeds, run it on the lower half; if it fails, run it on the upper half. At the end we have determined a value k_0 for which the algorithm succeeds while failing for $k_0 - 1$. Then, k_0 must be a lower bound on the chromatic number of the graph, and our performance no worse than if this was given.

Corollary 2.2 *The performance ratio of* Color *is $O(n/\log n)$.*

Proof. The ratio of the size of the approximation to the chromatic number χ of the graph is $O(n^{(\chi-2)/(\chi-1)}/\chi)$, which is maximized when $\chi \approx \log n$. ∎

3 Parallel Coloring Algorithm

In this section we discuss how to parallelize the steps of the preceding coloring algorithm, while maintaining an identical performance.

The main algorithm can be parallelized easily by executing the loop iterations in parallel. The bipartite graphs can be colored exactly by finding a forest of rooted spanning trees using the Euler tour technique (see [13]), computing the depth of each node using list ranking, and assigning nodes of odd depth one color and nodes of even depth the other color.

The hard part is finding a maximal partial coloring. A construction of Luby [16] reduces it to the maximal independent set problem which has known NC solutions.

We create a new graph G' that consists of copies of the original graph, with vertices in different copies adjacent iff they correspond to the same original vertex. Consider a maximal independent set I in G', and let C_i be the set of vertices in I belonging to the i-th copy of G in G'. Note that no two vertices in I correspond to the same vertex in G, thus I and C_i map one-to-one to subsets I' and C_i' of V. Moreover, each vertex in $V(G) - I'$ is adjacent to some

```
**MaximalPartialColor**$(G,x)$
begin
  Construct graph $G'$ on $n \cdot x$ vertices:
    $V(G') = \{v_i^j : v_i \in V(G),\ j = 1,\ldots,n\}$
    $E(G') = \{(v_i^j, v_y^z) : i = s \vee (j = z \wedge (v_i, v_y) \in E(G))\}$
  $I \leftarrow$ MIS$(G')$.
  $C_i \leftarrow I \cap G^i,\ i = 1,\ldots,x$
  ResidueNodes $\leftarrow G - \cup_{i=1}^{x} C_i$
  return $\{C_i\}$, ResidueNodes
end
```

Figure 2: Maximal partial coloring algorithm in parallel

vertex in each C_i', as otherwise its addition to I would still be independent. Hence, the C_i' form a maximal partial coloring of G.

The complexity of the maximal partial coloring algorithm is dominated by the complexity of the maximal independent set algorithm used. The faster algorithm known for the latter problem runs in $O(\log^2 |V|)$ time, due to Luby [16]. A processor-efficient algorithm of Goldberg and Spencer [7] runs in $O(\log^3 |V|)$ time with $(|V| + |E|)/\log |V|$ processors. The number of vertices of the composed graph G' is $s \cdot n$, where $s = s(n,k) = (n/(k-2))^{(k-2)/(k-1)}$ is the number of greedy bins allocated. The number of edges is then $s|E| + n\binom{s}{2} < s(|E| + ns)$, and thus the processor complexity of our algorithm would be $s(|E| + ns)/\log n < n^3$.

The complexity of the main algorithm is also dominated by k recursion levels of calls to the maximal partial coloring subroutine, adding a factor of $k = O(\log n)$ to the time complexity.

4 On-line Coloring Algorithm

We show in this section how to convert Color into an on-line algorithm.

The on-line coloring problem is defined as follows. An adversary selects a graph and an ordering of the vertices. It feeds us one node at a time along with its edges to the nodes already received. We are to irrevocably assign that node a color consistent with the colors of the previous nodes. We are allowed arbitrary amount of time but we are not given any further information about the graph, including its size and chromatic number.

In order to deter the adversary from foreseeing our every move, randomization frequently proves useful. An adversary is said to be *oblivious* if it picks the graph and the ordering before we make any choices (i.e. without knowledge of the values of the random bits), rather than constructing the graph adaptively according to our choices. We measure the expected number of colors used by a randomized algorithm on a given graph, where the expectation ranges over the values of the random bits. The performance ratio is then defined as the ratio of this expectation to the chromatic number, maximized over all graphs.

For some on-line problems, randomized algorithm against adaptive adversary form an intermediate model between the deterministic and the randomized oblivious cases. For the graph coloring problem, however, no results are known that are specific to this model.

In order to convert the off-line algorithm into an on-line one, several changes need to be made and hurdles to be overcome. The greatest change is that we must process the nodes fully in sequence, rather than processing the chromatic levels in sequence. We must also overcome three difficulties: the size of partitioning classes will not be known before they must be chosen, the size of the input will not be known, and the chromatic number will be unknown.

The first hurdle is that a partitioning class must be selected in advance, before we know its eventual size. The concern is that if the size will be large, many subproblems will be spanned. The solution is to use randomization: pick a uniformly random class as a partitioning class. Since the adversary must choose the ordering of the graph in advance, the expected number of nodes in the class will be sufficiently small.

The second problem is estimating the size of the input, and recursively, the sizes of the subproblems. For this we apply a well known trick in the on-line trade (see e.g. [15]), the *doubling strategy*, which works as follows. Assume, to begin with, that the size of the input, n, equals 2^i, for i initially equal 0. When the actual number of nodes exceeds this estimate, we increment the estimate to the next power of 2, retire the old color classes and start afresh with a new set of color classes. We continue this way and when the input finally ends our estimate of n is never off by more than a factor of 2. Since the function giving our approximation performance is necessarily convex, it follows that the overhead caused by this strategy is no more than a small constant.

It is important to notice that we can apply the same strategy recursively for the subproblems as well. From the perspective of an on-line algorithm, the subgraph induced by the nodes in a residue class (i.e. in a "subproblem") is structurally no different from the original problem, except for the fact that the chromatic number is one less. We shall therefore allocate sets of color classes to this subproblem completely independent of the other subproblems or the parent problem. This strategy is what primarily distinguishes our algorithm from Vishwanathan's.

The third problem – not knowing the chromatic number in advance – can be handled similarly. To begin with, we assume the graph is 2-colorable, and as our assumptions are shattered, usually by the bipartite algorithm stumbling upon a non-bipartite subgraph, we increment our estimate by one, and start from scratch with a fresh set of colors.

Finally, we need to find on-line versions of the subroutines called by algorithm K-Color. As stated in [15], there is a simple algorithm for on-line coloring bipartite graph using no more than $2\log n$ colors. And a First-Fit algorithm that is restricted to using only the greedy color classes actually available is perfect for performing a maximal partial coloring.

The resulting algorithm is shown in Figures 3-5.

```
**On-line-Color** ($G$)
{ On-line color a graph $G$ }
begin
  $k \leftarrow 2$
  $Root \leftarrow$ New-Coloring-Tree(2, $k$)
  foreach ($v \in V(G)$) in sequence do
     $Result \leftarrow$ K-Color-Online($Root$, $v$)
     if ($Result = Failure$) then
        $k \leftarrow k + 1$
        $Root \leftarrow$ New-Coloring-Tree(2, $k$)
end
```

Figure 3: On-line Graph Coloring Algorithm

The on-line algorithm colors the graph by constructing and maintaining a *coloring tree*. This tree contains each previous vertex in a *greedy color class* at some node in the tree. Each node in the tree contains the following static[1] information: n, the capacity of the subtree; k, the colorability of the subtree; $s = s(n,k)$, the number of greedy bins; and *part*, the index of the

[1]Static in the sense that the values are given at the creation of the subtree rooted by the node.

```
K-Color-Online ($T$, $v$)
{ Re-entrant procedure. }
{ Assign a color to the vertex $v$ using coloring tree $T$. }
{ Return success or failure }
begin
  with $T$ do
     if ($k \leq 2$) then
        return (BipartiteColor($T$, $v$))
     if ($N \geq n$) then
        $T \leftarrow$ New-Coloring-Tree($2n$, $k$)
     $N \leftarrow N + 1$
     for $j \leftarrow 1$ to $s$ do
        if ($v$ is non-adjacent to all vertices in $P[j]$) then
           Assign $v$ the color $P[j]$ (or $COLOR[v] \leftarrow P[j]$)
           return ($Success$)
     { $v$ must now be adjacent to some node in each color class }
     Non-deterministically choose a neighbor $w$ of $v$ in partitioning class $P[part]$
     if ($R[w]$ is uninitialized) then
        $R[w] \leftarrow$ New-Coloring-Tree($1$, $k-1$)
     return (K-Color-Online($R[w]$,$v$))
end
```

Figure 4: A randomized on-line coloring algorithm

```
New-Coloring-Tree ($n_0$, $k_0$)
{ Create and return a new coloring tree with capacity $n$ and chromatic upper bound $k$ }
begin
  with $T$ do
     $n \leftarrow n_0$
     $k \leftarrow k_0$
     $s \leftarrow s(n,k) = (2^c\ n/(k-2))^{(k-2)/(k-1)}$
     $P[1,\ldots,s] \leftarrow$ A new set of colors
     $part \leftarrow$ Random integer between $1$ and $s$
     $N \leftarrow 0$
  return $T$
end
```

Figure 5: Create and initialize a new coloring tree

partitioning bin. Each node also contains the following variables: N, the current number of vertices in the subtree; $P[1 \ldots s]$, the greedy bins (the bins themselves are static; their contents are dynamic); and, R_w, a *residue tree* for each vertex assigned to the partitioning bin $P[part]$. Finally, each node implicitly contains a host of cut-off subtrees.

Here, c is a constant, whose optimal value is determined in the analysis.

When a vertex v arrives, the algorithm will assign it to one of the static greedy bins at the root whenever possible. If this fails, we choose any neighbor w of v in the partitioning bin $P[part]$. The vertex is the sent recursively to the subproblem corresponding to the subtree $R[w]$ (which is constructed if necessary). This continues until the colorability index of the tree is at most two, in which case we know how to on-line color the node efficiently.

Two invariants must be maintained in each subtree of the coloring tree:

1. $N \leq n$. That is, the number of vertices in the subtree at any time cannot exceed the capacity of the subtree.

2. The colorability index for the subtree rooted at node v must be at least one greater than the colorability index for the subtree rooted at some child of v. This insures that the colorability index stays an upper bound on the chromatic number of the graph.

The former invariant is maintained by K-Color-Online by creating a new subtree with double the capacity, whenever the vertex count exceeds capacity. The old greedy bins (color classes) are then set aside and never used again, and the vertices they contain do not factor into the coloring assignment other than that their number is still a part of the content of the subtree.

The latter invariant is maintained by On-line-Color, again by cutting off the old tree and creating a new tree with a colorability index one larger. Such an action is triggered by the failure of the on-line bipartite coloring algorithm at some leaf, when some leaf becomes three-chromatic, and the effect filters up the path to the root.

4.1 Analysis

The analysis of the performance of the on-line algorithm resembles that of the parallel approximation, but is more involved. The heart of the analysis is contained in the following result.

Theorem 4.1 K-Color-Online *uses at most $O(n^{(k-2)/(k-1)}(\log n)^{1/(k-1)})$ expected number of colors on k-colorable graphs with n vertices.*

Proof. Let us first define some notation. Let $A(n,k)$ denote the maximum expected number of colors the algorithm uses to color any k-colorable graph on n vertices, when n is known in advance. Let $B(n,k)$ similarly be the maximum expected number of colors when n is not known in advance. Let c denote a real-valued constant to be defined later. We shall need the following functions and constants:

$$c_k = (\tfrac{2^c}{k-2})^{(k-2)/(k-1)}$$
$$s(n,k) = c_k \cdot n^{(k-2)/(k-1)}(\log n)^{1/(k-1)}$$
$$\mathcal{A}(n,k) = (k-1)s(n,k)$$
$$\mathcal{B}(n,k) = 2^{c/(k-1)}\mathcal{A}(n,k)$$

$\mathcal{B}(n,k)$ is the upper bound that we shall prove for $B(n,k)$, and similarly $\mathcal{A}(n,k)$ is the bound for $A(n,k)$. $s(n,k)$ denotes the number of greedy bins allocated, as previously defined. Default logarithm is base 2.

Given the above notation, the proposition of the theorem can be restated as

$$B(n,k) \le \mathcal{B}(n,k), \quad \text{for all } n,k,k \ge 2.$$

The proof proceeds by induction on k simultaneously for A and B. For the base case, observe that $B(n,2) = A(n,2) \le 2\log n = \mathcal{A}(n,2) = \mathcal{B}(n,2)$. To prove the inductive step, we first show that $A(n,k) \le \mathcal{A}(n,k)$, for any positive integer n.

The number $A(n,k)$ of bins used during the time that the size estimate n remains unchanged, equals the number of greedy bins plus at most the average cost of the subproblems over the s different choices of a partitioning bin. The latter amounts to at most

$$\frac{1}{s}\sum_{i=1}^{s}\sum_{j=1}^{n_i} B(m_j^{(i)}, k-1), \tag{1}$$

where n_i is the number of elements in greedy bin i and $m_j^{(i)}$ the size of the subgraph. Let g denote the total number of elements placed in greedy bins. By the inductive assumption and the convexity of $\mathcal{B}$, (1) is at most

$$\frac{1}{s}\max_g\{g \cdot \mathcal{B}(s \cdot \frac{n-g}{g}, k-1)\}.$$

We now plug in the value of $\mathcal{B}$, obtaining

$$\max_g\{g \cdot (\frac{n-g}{g})^{(k-3)/(k-2)}\} \cdot c_{k-1} \cdot (k-2) \cdot (2^c/s)^{1/(k-2)} \cdot (\log n)^{1/(k-2)}.$$

As in Theorem 1, the maximum of the inner function simplifies the formula, leading to

$$A(n,k) \;\le\; s + (k-2) \cdot s(n,k) \;=\; \mathcal{A}(n,k).$$

Hence, $A(n,k) \le \mathcal{A}(n,k)$, for any n. We now show that this implies that the same holds for B and $\mathcal{B}$, from which the theorem follows.

In general, we do not know the number of nodes in advance, at any level in the recursion. The algorithm fixes a value for n, and when that count has been exceeded, it doubles its expectation for the total number of vertices. We obtain $d = \lceil \log n \rceil$ subproblems, coloring the nodes $\{1\}, \{2,3\}, \ldots, \{2^{d-2}, \ldots, 2^{d-1}-1\}, \{2^{d-1}, \ldots, n\}$, respectively. Let rn denote the total number of nodes colored by the end of the last doubling step, i.e. $rn = 2^{d-1}$.

The total number of bins used is at most the sum of those used on each of the size-bounded subproblems, the residue bins used on the last $(1-r)n$ nodes, and the greedy bins allocated in the last step. That is

$$B(n,k) \;\le\; \sum_{i=1}^{d-1} \mathcal{A}(\frac{rn}{2^i}, k) + \mathcal{A}((1-r)n, k) + \frac{1}{k-1}\mathcal{A}(n,k).$$

The first term evaluates to

$$\mathcal{A}(rn,k)\sum_{i=1}^{d-1} 2^{-i(k-2)/(k-1)} \;\le\; \mathcal{A}(rn,k)/(2^{(k-2)/(k-1)} - 1).$$

Maximizing the first two terms over the choices for r using standard calculus yields

$$B(n,k) \;\le\; [(1 + (2^{(k-1)/(k-2)} - 1)^{-(k-1)})^{1/(k-1)} + \frac{1}{k-1}]\mathcal{A}(n,k).$$

Inspection shows that this is at most

$$B(n,k) \;\le\; 2^{3.77/(k-1)}\mathcal{A}(n,k).$$

Thus we set $c = 3.77$, and the theorem follows. ∎

Theorem 4.2 *Our on-line algorithm uses at most $O(n^{(\chi-2)/(\chi-1)}(\log n)^{1/(\chi-1)})$ colors, without* a priori *knowledge of either n or the chromatic number $\chi(G)$.*

Proof. We maintain a lower bounded estimate k on the chromatic number of the graph. This estimate is incremented only when the current graph has been proven to be non k-colorable. Consider the nodes colored while the estimate was i. If we denote their count by N_i, we know that On-line-Color used no more than $\mathcal{B}(N_i, i)$ colors to color those nodes. This can be bounded from above by $\mathcal{B}(N_i, k)$, which is a convex function. Thus the sum of these values, $i = 2, \dots, k$, is maximized when all N_i are equal, for a total of

$$(k-1)\ \mathcal{B}(\frac{n}{k-1}, k)\ \leq\ (k-1)^{1/(k-1)}\ \mathcal{B}(n,k)\ \leq 39 n^{(k-2)/(k-1)}(\log n)^{1/(k-1)}$$

number of colors. ∎

We obtain the same performance ratio as in the off-line case, with a similar proof.

Corollary 4.3 *Randomized on-line coloring is $O(n/\log n)$ approximable.*

5 Independent Sets in Parallel

The parallel coloring algorithm can also be used to find cliques or independent sets in parallel, via an observation due to Wigderson [18]. Recall that the algorithm used at most $n^{(k-2)/(k-1)}$ colors, where the graph was provably not $k-1$ colorable since a k-chromatic subgraph had been found. This subgraph is formed by the pivot nodes w_i at each of the $k-2$ recursive levels, along with a non-bipartite subgraph (i. e. an odd cycle) at the end of the recursion. These pivot nodes are, by definition, mutually adjacent, as well as adjacent to the odd cycle, thus producing a k-clique. Since the clique number of the graph is at most the chromatic number, which is at most the number of colors we used, this yields a $n^{(k-2)/(k-1)}/k$ approximation of the maximum clique, for an $O(n/\log n)$ performance ratio. Finally, recall that we can obtain an independent set algorithm with the same performance by applying our algorithm to the complement graph. Notice that the algorithm runs in polylogarithmic time, for all "interesting" values of k (i. e. $k \leq \log^2 n$).

Another source of a parallel independent set algorithm can be found in the first step of Berger and Rompel's [2] coloring algorithm. It essentially involves choosing polynomial number of random vertex subsets of size $\log_k n$ and testing for non-adjacency. This can be made deterministic via the pigeonhole principle without destroying the independence of the samples, and hence can be parallelized trivially. This method yields a $n/\log_k n$ approximation.

Notice that the two methods complement each other, since our/Wigderson's method performs best for small values of k, while the above random sampling performs relatively better for larger values of k. If we run both methods and retain the better result, we get an algorithm with a performance ratio of

$$\max_k \min\left(\frac{n^{(k-2)/(k-1)}}{k}, \frac{n}{\log_k n}\right)$$

which is minimized when $k = \log n / 2 \log\log n$ as $O(n(\log\log n/\log n)^2)$.

5.1 Graphs with large independent sets

We shall show how given a graph with an independence number at least $(1/3+\epsilon)n$, we can find an independent set of size $\Omega(\sqrt{n})$ in polylogarithmic time.

The method starts by removing a maximal collection of disjoint triangles (cliques of size 3) from the graph. Such a collection can be obtained as a maximal independent set in the graph whose vertices are all the triangles in the original graph, with edges between intersecting triangles. Since at most one node in each triangle can be a member of a given independent set, at least $\epsilon n = \Omega(n)$ nodes remain after the triangle removal.

```
**TriangleFreeIS** $(G)$
begin
  $I \leftarrow$ MaximalIndependentSet$(G) = (w_1, w_2, \ldots, w_t)$
  $\overline{N}_0 \leftarrow V(G)$
  for $i \leftarrow 1$ to $t$ do
    $\overline{N}_i \leftarrow \overline{N}_{i-1} \cap \{v \in V(G) : (v, w_i) \notin E(G)\}$
    $N_i \leftarrow \overline{N}_{i-1} \cap \{v \in V(G) : (v, w_i) \in E(G)\}$
  return $\max(N_1, \{w_1\} \cup N_2, \ldots, \{w_1, w_2, \ldots, w_{t-1}\} \cup N_t, \{w_1, w_2, \ldots, w_t\})$
end
```

Figure 6: Algorithm for finding independent sets in triangle free graphs.

Given a triangle-free graph, we apply the algorithm of Figure 6, which is based on a sequential algorithm of [3]. The algorithm finds a maximal independent set with an ordering of the vertices in the set, and finds the common non-neighborhoods $\overline{N}_i$ of any prefix of the vertices, as well as the complementary neighborhoods N_i. Each neighborhood must be independent since the graph contains no triangles, and is also contained in an earlier non-neighborhood. The final result is the largest of these independent sets considered, and its size must be at least the smallest t such that $\binom{t}{2} \geq n$, or approximately $\sqrt{2n}$.

Finding the recursive non-neighborhoods $\overline{N}_i$ is a prefix computation that can be performed in logarithmic time with linear number of processors. The complexity is therefore asymptotically equivalent to that of the MIS algorithm used.

The above construction can be generalized to find independent sets of size $\Omega(n^{1/(k-1)})$ in k-clique free graphs, providing a constructive parallel proof of the upper bound on Ramsey numbers by Erdős and Szekeres [4]. That also allows us to find such independent sets in graphs with independence number greater than n/k for $k > 3$, but, alas, the processor complexity of removing the k-cliques grows as fast as n^k.

Acknowledgements.

I would like to thank Ravi Boppana and Sundar Vishwanathan for helpful discussions, and the two anonymous referees for comments that improved the presentation. I also thank Sundar for re-inventing Luby's construction for me.

References

[1] D. Bean. Effective coloration. *J. Symbolic Logic*, 41:469–480, 1976.

[2] B. Berger and J. Rompel. A better performance guarantee for approximate graph coloring. *Algorithmica*, 5(4):459–466, 1990.

[3] R. B. Boppana and M. M. Halldórsson. Approximating maximum independent sets by excluding subgraphs. *BIT*, 32(2):180–196, June 1992.

[4] P. Erdős and G. Szekeres. A combinatorial problem in geometry. *Compositio Math.*, 2:463–470, 1935.

[5] U. Feige and J. Kilian. Zero knowledge and the chromatic number. In *Proc. of Conference on Computational Complexity*, June 1996.

[6] A. V. Goldberg, S. A. Plotkin, and G. E. Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM J. Disc. Math.*, 1(4):434–446, Nov. 1988.

[7] M. Goldberg and T. Spencer. Constructing a maximal independent set in parallel. *SIAM J. Disc. Math.*, 2(3):322–328, Aug. 1989.

[8] A. Gyárfás and J. Lehel. On-line and first fit colorings of graphs. *J. Graph Theory*, 12(2):217–227, 1988.

[9] P. Hajnal and E. Szemerédi. Brooks coloring in parallel. *SIAM J. Disc. Math.*, 3(1):74–80, 1990.

[10] M. M. Halldórsson. A still better performance guarantee for approximate graph coloring. *Inform. Process. Lett.*, 45:19–23, 25 January 1993.

[11] M. M. Halldórsson and M. Szegedy. Lower bounds for on-line graph coloring. *Theoretical Comput. Sci.*, 130:163–174, Aug. 1994.

[12] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. Manuscript, 1996.

[13] R. Karp and V. Ramachandran. A survey of parallel algorithms for shared-memory machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 17, pages 869–941. Elsevier Science Publishers B.V., 1990.

[14] H. A. Kierstead and W. T. Trotter. An extremal problem in recursive combinatorics. In *Proc. 12th Southeastern Conf. on Combinatorics, Graph Theory, and Computing. Congressus Numerantium XXXIII*, pages 143–153, 1981.

[15] L. Lovász, M. Saks, and W. T. Trotter. An online graph coloring algorithm with sublinear performance ratio. *Discrete Math.*, 75:319–325, 1989.

[16] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15:1036–1053, 1986.

[17] S. Vishwanathan. Randomized online graph coloring. *J. Algorithms*, 13:657–669, Dec. 1992.

[18] A. Wigderson. Personal communications.

[19] A. Wigderson. Improving the performance guarantee for approximate graph coloring. *J. ACM*, 30(4):729–735, 1983.