Edge-coloring bipartite graphs

Ajai Kapoor, Romeo Rizzi *

November 10, 1997

Abstract

Given a bipartite graph G with n nodes, m edges and maximum degree Δ, we find an edge coloring for G using Δ colors in time $T + \mathcal{O}(m \log \Delta)$, where T is the time needed to find a perfect matching in a k-regular bipartite graph with at most $\mathcal{O}(m)$ edges and $k \leq \Delta$. Together with best known bounds for T this implies an $\mathcal{O}((m + \frac{m}{\Delta} \log \frac{m}{\Delta} \log \Delta) \log \Delta)$ edge-coloring algorithm which improves on the $\mathcal{O}((m + \frac{m}{\Delta} \log \frac{m}{\Delta} \log^2 \Delta) \log \Delta)$ algorithm of Hopcroft and Cole. Our algorithm can also be used to find a $(\Delta + 2)$-edge-coloring for G in time $\mathcal{O}(m \log \Delta)$. The previous best approximation algorithm with the same time bound needed $\Delta + \log \Delta$ colors.

1 Introduction

An edge-coloring of a graph consists of an assignment of colors to its edges so that no adjacent edges receive the same color. A k-edge-coloring is one that uses at most k colors. Kőnig, in [4], showed that every bipartite graph G has a Δ-edge-coloring, where Δ is the maximum degree of a node in G.

Kőnig's proof yields an $\mathcal{O}(nm)$ algorithm, where n is the number of nodes of G and m the number of edges. Hopcroft and Cole, in [1], obtain an $\mathcal{O}((m + \frac{m}{\Delta} \log \frac{m}{\Delta} \log^2 \Delta) \log \Delta)$ algorithm. Recently, Schrijver [5] revisited the problem and obtained an $\mathcal{O}(m\Delta)$ algorithm. He also discussed how to obtain fast edge-coloring schemes in case the prime factors of n were small. In the same paper Schrijver left open the problem of finding an $\mathcal{O}(m \log \Delta)$ algorithm for bipartite edge-coloring. In this paper we provide an algorithm to Δ-edge-color a bipartite graph in time $T + \mathcal{O}(m \log \Delta)$, where T is the time needed to find a perfect matching in a k-regular bipartite graph with $\mathcal{O}(m)$ edges and $k \leq \Delta$.

Hopcroft and Cole, in [1], give an $\mathcal{O}(m + \frac{m}{\Delta} \log(\frac{m}{\Delta}) \log^2 \Delta)$ algorithm to find a perfect matching in a Δ-regular bipartite graph. Consequently, our edge-coloring algorithm has a time complexity of $\mathcal{O}((m + \frac{m}{\Delta} \log \frac{m}{\Delta} \log \Delta) \log \Delta)$, which improves on the best known time bound for bipartite edge-coloring held until now by Hopcroft and Cole's algorithm. Depending on the values of n and Δ, the fastest method to edge-color a bipartite graph is either the algorithm here introduced or the scheme of Schrijver.

In the next section we describe the algorithm and prove its time complexity. We also show how the algorithm can be used to obtain a $(\Delta + 2)$-edge-coloring in time $\mathcal{O}(m \log \Delta)$.

*Dipartimento di Matematica Pura ed Applicata, Università di Padova, Via Belzoni 7, 35131 Padova, Italy.
Ajai Kapoor was supported in part by a grant from Gruppo Nazionale Delle Ricerche-CNR.

2 Algorithm

Given a bipartite graph G, with m edges and maximum degree Δ, the following construction due to Schrijver [5] makes a Δ-regular graph $\overline{G}$ with at most $\mathcal{O}(m)$ edges so that an edge-coloring of $\overline{G}$ implies one for G. By first identifying nodes on the same side of the bipartition of G and with sum of the degrees $\leq \Delta$ we can always assume that G contains at most two nodes with degree $\leq \lfloor \frac{\Delta}{2} \rfloor$. Now a copy G' of G is added and $\Delta - d(v)$ parallel edges are added between node v and its copy in G'. The bipartite graph $\overline{G}$ so obtained is Δ-regular and has $\mathcal{O}(m)$ edges. Moreover a k-edge-coloring of $\overline{G}$ contains a k-edge-coloring of G. Keeping the above construction in mind we assume in the remainder of the paper that G is Δ-regular. Moreover we assume that $\Delta \geq 3$, since otherwise an edge coloring can be found in $\mathcal{O}(m)$.

An ordered list of positive integers is called a *bin*. Let $E_1, \ldots, E_p$ be a partition of $E(G)$ such that $G(V, E_i)$ is r_i-regular for every $i = 1, \ldots, p$. We associate to such a partition the bin $(r_1, \ldots, r_p)$. To the input graph G we associate the bin (Δ). To *solve* bin (Δ) means to transform (Δ) into another bin whose elements are all 1's by applying the following three elementary operations to the integers contained in the bin.

Operation SLICE-ONE (k) *require:* $k > 1$.
return $(1, k-1)$

Operation SPLIT-EVEN (k) *require:* k even.
return $(\frac{k}{2}, \frac{k}{2})$

Operation SPLIT-ODD (k_1, k_2) *require:* k_1, k_2 odd.
return *Split-even*$(k_1 + k_2)$

Slice-one(k) corresponds to finding a perfect matching in a k-regular bipartite graph. We denote the cost of this operation by $f(k; n)$.

Split-even(k), where k is even, corresponds to the following operation: Given a k-regular bipartite graph G, obtain two $\frac{k}{2}$-regular bipartite graphs G_1 and G_2, by taking alternate edges of an Euler tour of G, in G_1 and G_2. The cost of this operation is $\mathcal{O}(kn)$. The operation was introduced by Gabow in [2]. It was also extensively used in [3], [1] and [5].

Split-odd(k_1, k_2), where k_1 and k_2 are odd, corresponds to adding a k_1-regular bipartite graph to a k_2-regular bipartite graph and then executing a *Split-even*$(k_1 + k_2)$. The cost of this operation is $\mathcal{O}((k_1 + k_2)n)$.

In the following we give a method to solve bin (Δ). This method requires at most a single call to *Slice-one* and has a total cost of $f(k; n) + \mathcal{O}(\Delta n \log \Delta)$, where $k \leq \Delta$. A first consequence is a $T + \mathcal{O}(m \log \Delta)$ edge-coloring algorithm, where T is the time needed to solve a perfect matching problem in a k-regular bipartite graph with at most $\mathcal{O}(m)$ edges and $k \leq \Delta$.

We say that a bin $(b_1, \ldots, b_p)$ is *almost solved* if

i) $b_1 = 1$,

ii) $b_j \leq \sum_{i=1}^{j-1} b_i$ (for every $j \in \{2, \ldots, p\}$).

To solve an almost solved bin, iteratively solve a bin of the form $(1, \ldots, 1, b_j)$ containing at least b_j ones. This can be done in $\mathcal{O}(b_j n \log b_j)$. Therefore the total cost to solve an almost solved bin $(b_1, \ldots, b_p)$ is $\mathcal{O}(\sum_{i=1}^{p} b_i n \log b_i)$, which is $\mathcal{O}(\Delta n \log \Delta)$. So it is enough to transform the initial bin (Δ) into an almost solved bin.

We need some notation when dealing with bins. Let H and T be bins then (H, T) is the bin consisting of the elements of H followed by the elements of T. Similarly (a, A) is obtained by appending integer a to the start of bin A. Denote by $val(H)$ the sum of the integers in bin H.

A bin of the form $B = (H, T)$ is said *normal* when $H = (a, b, b)$ with a odd and $\gcd(a, b) = 1$, and T possibly empty but satisfying the following condition:

$$t_j \leq val(H) + \sum_{i<j} t_i \qquad \text{(for every } t_j \in T\text{)}. \tag{1}$$

Note that a normal bin with $a = b$ is almost solved. The following procedure transforms a bin (Δ) with $\Delta \geq 3$ into a normal bin.

Procedure 1 NORMALIZE (Δ) *require:* $\Delta \geq 3$

If Δ is odd, then by *Slice-one*(Δ) obtain the bin $(1, \Delta - 1)$. By *Split-even*$(\Delta - 1)$ obtain the normal bin $(1, \frac{\Delta-1}{2}, \frac{\Delta-1}{2})$.

If Δ is even, then by *Split-even*(Δ) obtain the bin $(\frac{\Delta}{2}, \frac{\Delta}{2})$. If $\frac{\Delta}{2} = 2$ then apply *Split-even* twice more to obtain the normal bin $(1, 1, 1, 1)$. Otherwise $\frac{\Delta}{2} \geq 3$ and by recursion $(Normalize(\frac{\Delta}{2}), \frac{\Delta}{2})$ is a normal bin.

The cost of $Normalize(\Delta)$ is at most $f(k; n) + \mathcal{O}(\Delta n)$, where $k \leq \Delta$.

An important ingredient of our solution is the following procedure:

Procedure 2 HIT-EVEN (a, b, b) *require:* a and b odd and $a \neq b$

If $\frac{a+b}{2}$ is even return $(b$,*Split-odd*$(a, b))$. Else return *Hit-even*$(b$, *Split-odd*$(a, b))$.

Hit-even transforms a bin $H = (a, b, b)$ with a and b distinct and odd into a bin $H' = (a', b', b')$ where a' is odd and b' is even, while $val(H') = val(H)$. The cost of *Hit-even*(a, b, b) is denoted by $T(a, b, b)$ and is given by the recursion $T(a, b, b) = (a + b)n + T(b, \frac{a+b}{2}, \frac{a+b}{2})$. Note that in the bin $(b, \frac{a+b}{2}, \frac{a+b}{2})$ the absolute value of the difference between the two distinct integers is $|\frac{b-a}{2}|$, which is half of $|b - a|$. Thus within at most $\log(|b - a|)$ recursive calls the procedure terminates. Since $|b - a| < \Delta$, then $T(a, b, b) \leq (a + 2b)n \log \Delta$.

Observation 1: If (H, T) is a normal bin then $($*Hit-even*$(H), T)$ is a normal bin.
Proof: Inside *Hit-even*, when *Split-odd* is applied to a bin $H = (a, b, b)$ a second bin $H' = (a', b', b') = (b, \frac{a+b}{2}, \frac{a+b}{2})$ is obtained. Note that $\gcd(a', b')$ divides $a' = b$ and $b' = \frac{a+b}{2}$, hence $a + b$ and a. Thus $\gcd(a', b') \leq \gcd(a, b) = 1$. Moreover *Hit-even* does not modify $val(H)$. $\square$

Finally the following procedure transforms the initial bin (Δ) into an almost solved bin.

Procedure 3 ALMOST-SOLVE (Δ)

$(H,T) \leftarrow Normalize(\Delta)$; *comment:* consider $H = (a,b,b)$.
loop 1 : *while* $(a \neq b)$ *do*
  loop 2 : *while* $(b$ *is even*$)$ *do*
    $T \leftarrow (b,T)$; *comment:* append one b to the beginning of T
    $H \leftarrow (a,$*Split-even*$(b))$; *comment:* *Split-even* the other b to create new H
  end while
  if $(a \neq b)$ *then* $H \leftarrow$*Hit-even*(H);
end while
return (H,T).

Lemma 1: *Almost-solve*(Δ) returns an almost solved bin (H,T).
Proof: At the start, procedure *Normalize* returns a normal bin (H,T). We show that the bin (H,T) remains normal throughout procedure *Almost-solve*. Consequently, when the procedure halts, then $a = b$ and the bin is almost solved.

In *loop 2*, when a b moves from H to the beginning of T, then for the new (H,T) $val(H) = a + b \geq b$ and condition 1 still holds. Since $\gcd(a, \frac{b}{2}) \leq \gcd(a,b)$, then (H,T) is normal after the application of *loop 2*, if it is normal before.

Observation 1 completes the proof. □

Lemma 2: In between two calls to *Hit-even*, $val(H)$ decreases at least by a factor of $\frac{3}{4}$.
Proof: Assume *Hit-even* is called with $H = (a,b,b)$ and returns bin (a',b',b'). At the next call of *Hit-even* , $val(H)$ is at most $a' + b'$. In case $a < b$, then $b' \geq \frac{a+b}{2} > \frac{val(H)}{4}$ and the claim follows. In case $a > b$, then either $b' = \frac{a+b}{2}$ and $b' > \frac{val(H)}{4}$ as above or *Hit-even* is called recursively on $(b, \frac{a+b}{2}, \frac{a+b}{2})$ and since $b < \frac{a+b}{2}$ the previous case applies. □

Lemma 3: *Almost-solve*(Δ) costs $f(k;n) + \mathcal{O}(\Delta n \log \Delta)$, where $k \leq \Delta$.
Proof: We first show that the procedure terminates. By Lemma 2, in every iteration of *loop 1* $val(H)$ decreases by a factor of $\frac{3}{4}$, unless procedure *Hit-even* is not called. When procedure *Hit-even* is not called then $a = b$ and the procedure terminates.

The cost of $Normalize(\Delta)$ is $f(k;n) + \mathcal{O}(\Delta n)$, where $k \leq \Delta$.

The cost of a single call to *Hit-even*(H) is $\mathcal{O}(val(H) n \log \Delta)$. By Lemma 2, between two calls to *Hit-even* the $val(H)$ decreases at least by a factor of $\frac{3}{4}$. Thus the total cost of *Hit-even* is $\mathcal{O}(\Delta n \log \Delta + \frac{3\Delta}{4} n \log \Delta + \ldots)$, which is $\mathcal{O}(\Delta n \log \Delta)$.

The cost of a single application of *loop 2* is $\mathcal{O}((v - v')n)$, where $v = val(H)$ at the start of the loop and $v' = val(H)$ at the termination. At the beginning of *loop 1* $val(H) \leq \Delta$. At the end $val(H) = 3$. Hence the total cost of *loop 2* is $\mathcal{O}(\Delta n)$. □

Approximation algorithm

Assume we are given a Δ-regular bipartite graph G with $\Delta \geq 3$. To $(\Delta+2)$-edge-color G in $\mathcal{O}(m \log \Delta)$ time we redefine the procedure *Normalize*.

Procedure 4 NORMALIZE (Δ)

If Δ is even, obtain $(\frac{\Delta}{2}, \frac{\Delta}{2})$ by *Split-even*(Δ). Add an arbitrary perfect matching to G and *return* $(1, \frac{\Delta}{2}, \frac{\Delta}{2})$.

If Δ is odd, then add an arbitrary perfect matching to G. Apply *Split-even*($\Delta+1$). Add a second arbitrary perfect matching and *return* $(1, \frac{\Delta+1}{2}, \frac{\Delta+1}{2})$.

After this procedure *Normalize*, our algorithm edge-colors a $[(\Delta+1)$ *or* $(\Delta+2)]$-regular bipartite graph containing G in time $\mathcal{O}(m \log \Delta)$. The edge-coloring obtained contains a $(\Delta+2)$-edge-coloring for G.

3 Conclusion

The edge-coloring algorithm for bipartite graphs, presented in this paper, has the best known time complexity. Furthermore the $(\Delta+2)$-edge-coloring approximation in time $\mathcal{O}(m \log \Delta)$, is better than the $(\Delta + \log \Delta)$-edge-coloring approximation in [5].

It is clear from our result that finding an $\mathcal{O}(m \log \Delta)$ edge-coloring algorithm for bipartite graphs is equivalent to finding a perfect matching in a Δ-regular bipartite graph in time $\mathcal{O}(m \log \Delta)$. This equivalence was pointed out already by Schrijver in [5].

We would like to thank Prof. A. Schrijver for providing us with a manuscript of his paper. We would also like to thank Michele Bugliese and Prof. Gilberto File for taking part in a stimulating discussion, that was of great help in reaching the results in this paper.

References

[1] R. Cole, J. Hopcroft, On edge coloring bipartite graphs, *SIAM Journal on Computing* 11 (1982) 540-546.

[2] H. N. Gabow, Using Euler partitions to edge color bipartite multigraphs, *International J. Computer and Information Sciences* 5 (1976) 345-355.

[3] H. N. Gabow, O. Kariv, Algorithms for edge coloring bipartite graphs and multigraphs, *SIAM Journal on Computing* 11 (1982) 117-129.

[4] D. Kőnig, Graphok és alkalmazásuk a determinánsok és a halmazok elméletére [Hungarian], *Mathematikai és Természettudományi Értesito* 34 (1916) 104-119.

[5] A. Schrijver, Bipartite edge-colouring in $\mathcal{O}(\Delta m)$ time, preprint, CWI (1996).