Approximate Max k-Cut with Subgraph Guarantee

Viggo Kann* Jens Lagergren† Alessandro Panconesi‡

June 3, 1997

Abstract

We study the following variant of the Max k-Cut problem. Given an input graph G with positively weighted edges and k colors, we wish to compute a subgraph H of G containing "lots" of heavy edges and a color assignment $c : V \to [k]$ such that: (a) all edges in H are properly colored and (b) a "large fraction" of edges in $G \setminus H$ is properly colored. We give several definitions of "lots" and "large fraction" and give fast polynomial time algorithms to compute such color assignments. This problem is related to the frequency allocation problems for cellular telephone networks but could be useful in other scenarios too.

Key words: Algorithms; Approximation; Max k-Cut; Spanning subgraphs; Induced subgraphs

1 Introduction.

In the *frequency allocation problem for cellular telephones* we are given a set of k frequencies, henceforth referred to as *colors*, and a weighted input graph which intuitively models a network of radio transmitters. In the graph, vertices correspond to transmitters, and an edge (u, v) denotes the fact that if u and v get the same color (*i.e.* frequency) the performance locally deteriorates by $w(u, v)$ per cent. Usually, the set of colors is fixed once and for all and it is independent of the input graph. The goal is to allocate colors so to minimize the "total disruption", a notion that can be formalized in several ways. In many situations one is seeking to optimize simultaneously two somewhat conflicting parameters. On one hand, one would like to take care of "heavy" edges as much as possible because if even one of these were monochromatic, the resulting deterioration of local service would be unacceptable (an edge is *monochromatic* if both endpoints have the same color). On the other, one would like the color assignment to be good globally, *i.e.* the fraction of monochromatic edges should be small.

One way to formalize the problem is to introduce a parameter h and to define the heavy edges as those whose weight is greater than or equal to h. This formulation however immediately leads to an NP-hard problem because asking whether such a subgraph is k-colorable is an NP-complete problem. Therefore other approaches are called for. In this paper we propose two ways to define the graph of "heavy"

*Department of Numerical Analysis and Computing Science, Royal Institute of Technology, SE-100 44 Stockholm, Sweden. Email: viggo@nada.kth.se. Supported by grants from TFR.

†Department of Numerical Analysis and Computing Science, Royal Institute of Technology, SE-100 44 Stockholm, Sweden. Email: jensl@nada.kth.se. Supported by grants from NFR and TFR.

‡Informatik, Humboldt Universität, Unter den Linden 6, 10099 Berlin, Germany. Email: ale@informatik.hu-berlin.de. Supported by an Alexander von Humboldt research fellowship. This work was done when the author was visiting SICS and the Royal Institute of Technology in Stockholm, with financial support provided by an ERCIM post-doctoral fellowship.

edges which do not give raise to an NP-complete problem and that could work well in many real life situations. The basic idea is that we are given a subgraph H of G which: (a) is easily computable; (b) is easily k-colorable, and (c) contains lots of "heavy" edges (hopefully all). Our goal is then to compute a color assignment of G that satisfies these two conditions simultaneously: (1) no edge of H is monochromatic; (2) only a "small" fraction of the remaining edges is monochromatic.

In this paper we show how the above goal can be achieved for two categories of graphs: maximum l-matchings (an l-matching is a subgraph of degree at most l) and unions of maximum weight spanning trees. We will investigate the following scenarios.

- H is a maximum $(k-1)$-matching in G. In this case the fraction of monochromatic edges in $G \setminus H$ is at most $2\ln(k+1)/k$. This can be achieved in $O(|E|)$ probabilistic and $O(k^5|E| + k|V|)$ deterministic sequential time.
- H is a maximum $(k/2)$-matching in G. In this case the fraction of monochromatic edges in $G \setminus H$ is at most $2/k$. This can be achieved in $O(|E| + k|V|)$ deterministic sequential time.
- H is a maximum $(\log k - 1)$-matching in G. In this case the fraction of monochromatic edges in $G \setminus H$ is at most $1/k$. This can be achieved in $O(|V|\sqrt{|V|\log|V|}\log k)$ deterministic sequential time.
- H is the union of t maximum weight spanning trees (MST's), where $t \leq k/2$. In this case the fraction of monochromatic edges in $G \setminus H$ is at most $1/(k-2t+1)$. This can be achieved in $O(|E| + k|V|)$ deterministic sequential time.

How good are these solutions? To have an idea consider the MAX k-CUT problem: given an input graph G with weighted edges, find a color assignment that maximizes the total weight of non monochromatic edges. The best known approximation algorithm computes color assignments such that the fraction of monochromatic edges is $(1-o(1))/k$ [10, 3]. On the other hand, in a recent paper Kann et al. [8] show that no polynomial time algorithm can guarantee a fraction of monochromatic edges smaller than $\alpha/(2k)$, unless P=NP. Here α is the lower-bound for the approximability of MAX CUT; the current best value of α is 1/17 [7]. By means of straightforward reductions from MAX CUT, it is possible to show that the same lower bound applies to our problems (for which higher lower bounds could perhaps be found).

Our algorithms are randomized, but can be easily derandomized by standard applications of the method of conditional probabilities (see, for instance, [1, 9]). We assume that random numbers can be obtained for free.

2 Preliminaries

We will use the following notation.

If G is a graph, then $\Delta(G)$ denotes the maximum degree of any vertex of G.

The set $\{1, \ldots, n\}$ is denoted by $[n]$. A mapping $c : V \rightarrow [k]$ is called a *k-semi-coloring*.

We define a *maximum l-matching* in G to be a subgraph $H \subseteq G$ with vertex set $V(G)$ and $\Delta(H) \leq l$ that has maximum weight among such subgraphs.

A graph G is *d-inductive* if (1) it has a single vertex or (2) there is a vertex of degree at most d in G such that $G \setminus \{v\}$ is d-inductive.

Let H and H' be subgraphs of G. Then, $G \setminus H$ is the graph with vertex set $V(G)$ and edge set $E(G) \setminus E(H)$. Similarly, $H \cup H'$ is the graph with vertex set $V(G)$ and edge set $E(H) \cup E(H')$.

As customary, $\mathcal{H}(n) = \sum_{i=1}^{n} 1/i$ denotes the n-th harmonic number. We shall make use of the following simple fact.

Fact 1

$$\frac{1}{k+1}\sum_{i=1}^{k}\mathcal{H}(i) \;=\; \mathcal{H}(k+1)-1 \;<\; \ln(k+1).$$

This follows from the facts that $\sum_{i=1}^{k-1}\mathcal{H}(i) = k(\mathcal{H}(k)-1)$ and $\ln k < \mathcal{H}(k) < \ln k+1$ for all integers $k > 1$ (see for example [6]).

3 Maximum matchings

3.1 Many guaranteed edges

In this subsection we give an algorithm which, given a weighted input graph G and a maximum $(k-1)$-matching H in G, finds a k-semi-coloring $c : V \to [k]$ of G such that (a) no edge of H is monochromatic, and (b) at most a $2\ln(k+1)/k$ fraction of edges in $G \setminus H$ is monochromatic.

The only data structure needed by the algorithm is a list $A(u)$ of available colors which is initially set to $[k]$ for each vertex u. The input is G and H.

Algorithm A.

1. Let $|V| = n$ and $\sigma : V \to [n]$ be a random permutation of the vertices of G.
2. [Comment: the idea is to use σ as a *schedule* for the vertices in H to get their final color.]
   For $i = 1, 2 \ldots, n$ do: Let u be the vertex such that $\sigma(u) = i$; u picks a final color $\chi(u)$ uniformly at random from its current palette $A(u)$. The H-neighbors of u update their lists by removing $\chi(u)$.

In the next two lemmas, we establish correctness of the algorithm. Afterwards the running time is analyzed.

Lemma 2 *No edge of H is monochromatic.*

Proof: This follows from the way the lists are updated and the fact that they are never empty for vertices in H. □

Lemma 3 *For every edge e of $G \setminus H$, the probability that e is monochromatic is at most $2\ln(k+1)/k$.*

Proof: There are two kinds of edges: (a) edges of H; (b) edges of $G \setminus H$. We know from Lemma 2 that no edge of type (a) is monochromatic. For an edge $e = (u, v)$ of type (b) we compute the probability that it is monochromatic.

The random schedule σ induces a sub-permutation among each H-vertex u and its H-neighbors (if they exist); u can occupy every position within this sub-schedule with uniform probability. We refer to u's position within this sub-permutation as the *local time* of vertex u and denote it by $\ell(u)$. Suppose that the following information is known: (a) whether $\sigma(u) < \sigma(v)$ or $\sigma(u) > \sigma(v)$, i.e. which of these two vertices picks first, and (b) the local time of the second vertex. Then, assuming without loss of generality that $\sigma(u) < \sigma(v)$, we can upper bound the probability that $e = (u, v)$ is monochromatic by assuming pessimistically that: *(i)* the color picked by u is present in v's palette when v performs its choice, *(ii)* all H-neighbors of v who are scheduled before v pick different colors, and *(iii)* v has $k-1$ H-neighbors. In other words,

$$\Pr(e \text{ is monochromatic} \mid \sigma(u) < \sigma(v)) \leq \frac{1}{k-\ell(v)+1},$$

since v's palette at the time of its color choice has size at least $k - \ell(u) + 1$.

Consider then a set of $k+1$ elements containing both u and v and let π be a permutation of these elements chosen uniformly at random. The positions of u and v inside π determine who picks first, while the position of the second vertex determines its local time, i.e. if, say, $\pi(u) < \pi(v)$ then $\ell(v) = \pi(v) - 1$. Let $\pi(uv) = ij$ be a shorthand notation for $\pi(u) = i \wedge \pi(v) = j$. Notice that

$$\begin{aligned}
\Pr(\chi(u) = \chi(v) \quad & \mid \quad \pi(uv) = ij, i < j) \\
&= \sum_{c \in [k]} \Pr(\chi(u) = c \mid \pi(uv) = ij, i < j, \chi(u) = c) \Pr(\chi(u) = c) \\
&\leq \sum_{c \in [k]} \frac{1}{k - j + 2} \frac{1}{k} \\
&< \frac{1}{k - j + 1}.
\end{aligned}$$

Then, recalling Fact 1,

$$\begin{aligned}
\Pr(e \text{ is monochromatic}) &= \sum_{i \neq j} \Pr(\chi(u) = \chi(v) \mid \pi(uv) = ij) \Pr(\pi(uv) = ij) \\
&= 2 \sum_{i < j} \Pr(\chi(u) = \chi(v) \mid \pi(uv) = ij, i < j) \frac{1}{(k+1)k} \\
&< \frac{2}{(k+1)k} \sum_{i=1}^{k} \sum_{j=i+1}^{k+1} \frac{1}{k - j + 1} \\
&= \frac{2}{(k+1)k} \sum_{i=1}^{k} \mathcal{H}(i) = \frac{2\,(\mathcal{H}(k+1) - 1)}{k} \\
&< \frac{2\,\ln(k+1)}{k}.
\end{aligned}$$

□

We now turn to the running time of the algorithm.

Under the assumption that random numbers are given for free, step 1 requires $O(|V|)$ time. In step 2 we update the palettes of H-vertices. This takes $O(|E(H)|) = O(|E(G)|)$ time because when we update the palettes of the H-neighbors of a vertex u we can charge the edges incident on u. Altogether the algorithm requires $O(|E(G)|)$ time. The algorithm is probabilistic but can be derandomized with a standard application of the method of conditional probabilities (see for example [1, 9]).

How hard is it to find H? The problem of finding a maximum weight l-matching can be reduced to that of finding a maximum weighted matching [11], as follows. Replace each vertex v of degree d with $2d - l$ vertices $u_1, \ldots, u_d$ and $w_1, \ldots, w_{d-l}$. For each edge e incident to v, there is one vertex u_i that inherits e; however, no u_i inherits two edges. There is a complete bipartite graph between the vertices $u_1, \ldots, u_d$ and $w_1, \ldots, w_{d-l}$. Each edge in this complete bipartite graph has weight infinity. It is straightforward to verify the correctness of this reduction. The constructed graph has $O(|E|)$ vertices and $O(|V|^3)$ edges.

Maximum weighted matching in a graph with n vertices and m edges can be computed in time $O(n(m + n \log n))$ [4]. Thus H can be found in time $O(|V|^3|E|)$.
Remark: a drawback of this approach is the computation of H which can be quite expensive. Notice however that in some situation like frequency allocation the topology remains more or less constant, while the color allocation might change. In such cases one would compute H only once and repeated semi-color assignments would take only $O(|E|)$ time each.

3.2 Probability $1/k$ for non-guaranteed edges

Our first result in this subsection is the following: Given a graph and a maximum 1-matching H of G one can bicolor G such that all edges of H are bichromatic and at least half of the edges of $G \setminus H$ are bichromatic. We now give the algorithm.
Algorithm B.

1. Direct each edge of H arbitrarily.
2. For each directed edge (u, v) of H flip a fair coin and reverse its direction with probability 1/2. A vertex receives color 1 if the edge is pointing toward it, and color 0 otherwise.
3. Every vertex that is not adjacent to any edge in H picks color 0,1 with uniform probability.

Lemma 4 *After the execution of Algorithm B no edge of H is monochromatic. Also, for every edge e of $G \setminus H$, the probability that e is monochromatic is $1/2$.*

Proof: It is trivial to see that no edge of H is monochromatic. Note that for an edge (u, v) of $G \setminus H$ the colors of u and v are assigned independently. Hence, the probability that an edge of $G \setminus H$ is monochromatic is $1/2$. □

Algorithm B runs in $O(|V|)$ probabilistic time and the derandomized variation of it runs in $O(|V| + |E|)$ deterministic time.

The above result can be generalized to the following. Given a graph G and a maximum $(\lfloor \log k \rfloor - 1)$-matching H of G, G can be colored in such a way that no edge of H is monochromatic and at most a fraction $1/k$ of the edges of $G \setminus H$ is monochromatic. We now describe the algorithm. In the algorithm, each vertex v of G is colored with a vector $\chi(v) \in \{0, 1\}^{\lfloor \log k \rfloor}$.
Algorithm C.

1. Direct each edge of H arbitrarily.
2. Edge-color H using the colors in $[\lfloor \log k \rfloor]$
3. For each directed edge (u, v) of H, flip a fair coin and reverse its direction with probability 1/2.
4. If an edge with color i is incident on u, set the vector component $\chi(u)_i$ to 1 if the edge is pointing toward u, and set it to 0 otherwise. If no edge with color i is incident on u, set $x(u)_i$ to 1 or to 0 with probability 1/2.

Lemma 5 *After the execution of Algorithm C no edge of H is monochromatic. Also, for every edge e of $G \setminus H$, the probability that e is monochromatic is $1/k$.*

Proof: It is trivial to see that no edge of H is monochromatic. Note that for an edge (u, v) of $G \setminus H$ the colors of u and v are chosen independently and uniformly from among k colors. Hence, the probability that an edge of $G \setminus H$ is monochromatic is $1/k$. □

The existence of the edge coloring of step 2 is ensured by Vizing's Theorem (see, for instance, [2]). Such an edge coloring can be computed in $O(|E(H)|\sqrt{|V(H)| \log |V(H)|}) = O(|V(G)| \log k \sqrt{|V(G)| \log |V(G)|})$ time [5]. This bound dominates the complexity of the whole algorithm.

3.3 Probability $O(1/k)$ for non-guaranteed edges

In the next section, we will show that when H is a d-inductive graph, we can obtain probability $1/(k - d)$ that an edge of $G \setminus H$ is monochromatic. However, we already now note that this implies the following.

Lemma 6 *For each $0 < c < 1$ when H is a maximum ck-matching in G we can in polynomial time k-color G such that (1) no edge of H is monochromatic (2) the probability that an edge of $G \setminus H$ is monochromatic is $1/((1 - c)k)$.*

4 Inductive graphs

In this section, we show that when H is a d-inductive subgraph of G then G can be k-colored such that (1) no edge of H is monochromatic, (2) the probability that an edge of $G \setminus H$ is monochromatic is $1/(k-d)$. We will then apply this result to the case when H is a union of t trees by showing that every such union is $(2t-1)$-inductive.

4.1 Algorithm for d-inductive subgraphs

We are given G and a d-inductive subgraph H of G where $d < k$. The following algorithm colors G with k colors.

Algorithm D.

1. Delete a vertex v of degree d in H from H.
2. Color $H \setminus \{v\}$ recursively.
3. Color v with a color chosen uniformly from the colors not present in the neighborhood of v in H.
4. Each vertex $v \in G - H$ picks a color in $[k]$ independently and uniformly at random.

Lemma 7 *After the execution of the algorithm no edge of H is monochromatic. Also the probability that an edge of $G \setminus H$ is monochromatic is $1/(k-d)$.*

Proof: It is trivial to see that no edge of H is monochromatic. Let (u, v) be an edge of $G \setminus H$. If at least one of u and v is not in H then, the probability that (u, v) is monochromatic is $1/k$. If u and v are both in H we can assume pessimistically that the palettes coincide when they color. The probability that (u, v) is monochromatic is then at most $1/(k-d)$ because the two colors are chosen independently from two palettes of size $(k-d)$. $\square$

The algorithm runs in $O(k|V|)$ probabilistic time. Its derandomized variation runs in $O(|E| + k|V|)$ deterministic time.

4.2 A union of t trees

In this subsection, we show that a union of t trees is $(2t-1)$-inductive. Hence, when H is such a union we can k-color G such that (1) no edge of H is monochromatic (2) the probability that an edge of $G \setminus H$ is monochromatic is $1/(k-2t+1)$. Algorithm E shows that such an H is easy to find.

To choose a heavy union of t trees we will generate a sequence of graphs $G_1, G_2, \ldots, G_t, G_{t+1}$, a sequence of subgraphs $H_1, H_2, \ldots, H_t$, and a sequence of MSTs $T_1, T_2, \ldots, T_t$ such that: (a) T_i is a MST in G_i; (b) $G_1 = G$ and $G_{i+1} = G_i \setminus T_i$; and $H_1 = T_1$ and $H_{i+1} = H_i \cup T_{i+1}$. We will then let $H = H_t$.

Lemma 8 *A union of t trees is $2t$-colorable.*

Proof: Let H be a union of t forests. We make two observations. First, if v is a vertex of H then $H \setminus \{v\}$ is a union of at most t forests. Second, a union of t forests contains a vertex of degree $2t-1$. The latter holds since if the union contains n vertices then the degree sum in the union is at most $2t(n-1)$ and this implies that one vertex has degree at most $\lfloor (2t(n-1)/n) \rfloor = 2t-1$. From these two observations follows that H is $(2t-1)$-inductive; and this clearly implies that it is $2t$-colorable. $\square$

Algorithm E.

1. $G_1 := G$; $H_0 := \emptyset$

2. **for** $i := 1$ **to** t **do**
   Let T_i be a maximum spanning tree in G_i.
   $H_i := H_{i-1} \cup T_i$
   $G_{i+1} := G_i \setminus T_i$
3. $H := H_t$

How hard is it to find H? One maximum spanning tree can be constructed using Prim's algorithm in $O(|E| + |V| \log |V|)$ time. The set operations take $O(|E|)$ time. It follows that we can find H in $O(t(|E| + |V| \log |V|))$ deterministic time.

Acknowledgements

We would like to thank the anonymous referees for valuable suggestions and comments.

References

[1] N. Alon, J. Spencer and P. Erdös, *The Probabilistic Method*, John Wiley & Sons, Chichester, 1992.

[2] B. Bollobás, *Graph Theory: An Introductory Course*, Graduate Texts in Mathematics, Springer-Verlag, New York, 1979.

[3] A. Frieze and M. Jerrum, Improved approximation algorithms for MAX k-CUT and MAX BISECTION. In *Proc. of 4th Int. Conf. on Integer Prog. and Combinatorial Optimization*, Lecture Notes in Comput. Sci. 920, pages 1–13. Springer-Verlag, 1995.

[4] H. Gabow, Data structures for weighted matching and nearest common ancestors with linking. In *Proc. 1st Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 434–443, ACM-SIAM, 1990.

[5] H. Gabow, T. Nishizeki, O. Kariv, D. Leven, and O. Terada, Algorithms for edge coloring graphs. Tech. report TRECIS-8501, Tohoku Univ, 1985.

[6] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics*. Addison Wesley, Reading, Massachusetts, 1989.

[7] J. Håstad, Some optimal inapproximability results. In *Proc. 29th Ann. ACM Symp. on Theory of Comp.*, pages 1–10, ACM, 1997.

[8] V. Kann, S. Khanna, J. Lagergren and A. Panconesi, On the hardness of approximating MAX k-CUT and its dual. *Chicago J. Theoretical Comput. Sci.*, **1997**, number 2, 1997.

[9] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, 1995.

[10] C. H. Papadimitriou and M. Yannakakis, Optimization, approximation, and complexity classes. *J. Comput. System Sci.*, **43**, pages 425–440, 1991.

[11] W. T. Tutte, A short proof of the factor theorem for finite graphs. *Canadian J. Math.*, **6**, pages 347–352, 1954.