communications in mathematical
match
and in computer chemistry

no. 40, October 1999

ISSN 0340-6253

MATCDY (40) 265-272 (1999)

Computing equitable partitions of graphs

Oliver Bastert*
Zentrum Mathematik, Technische Universität München, Germany

Abstract

In this paper, we discuss a new algorithm for computing the coarsest equitable partition of a graph. The algorithm presented here is space optimal and fulfills the best known time bound of $O(m \log(n))$.

1 Introduction

In this paper, we discuss a new algorithm for computing the *coarsest equitable partition* of the vertex set of a graph. Equitable partitions were first introduced in [Sac66, Sac67, Sch74]. They play a crucial role in attacking the graph automorphism and the graph isomorphism problem (see [McK81]). For a general reference on equitable partitions see [CG97, CRS97, God93] and the contribution [ST99] in this volume.

The preferred way to illustrate our algorithm is to think of vertex-colored graphs, two vertices having the same color iff they lie in the same set of the partition. Hence, we will deal with vertex-colored graphs instead of graphs and vertex partitions.

The coarsest equitable partition, which is the most important one, is also known as the *total degree partition* This term indicates the idea of the algorithm presented in the following. It starts by coloring the vertices by their degrees, i.e., two vertices obtain the same color iff they have the same degree. In the next step, two vertices obtain the same color iff they had the same color before and, for each color occurring in the graph, they have the same number of neighbors of this color. In general, the number of colors will increase during this step. It is repeated until the number of colors does not increase anymore.

An efficient implementation of a higher dimensional variant of this algorithm, known as Weisfeiler-Lehman algorithm, has been discussed in [Bas98].

After introducing the basic notation, we present the ideas which allow the new algorithm to compute the coarsest equitable partition to require linear space and fulfill the

*Email: bastert@mathematik.tu-muenchen.de

Supported by the Deutsche Forschungsgemeinschaft through the graduate program Angewandte Algorithmische Mathematik, Technische Universität München.

best known time bound of $O(m\log(n))$. Finally, we discuss an implementation based on the presented ideas.

2 Basic notation

Let $G = (V, E, f)$ be a *vertex-colored graph*, i.e., G consists of a *vertex set* $V := \{1, 2, \dots, n\}$, an *edge set* $E \subseteq \mathbb{P}^2(V)$, $m := |E|$, and a *vertex-coloring*, $f : V \longrightarrow \{1, 2, \dots, n\}$. To simplify the discussion, we deal only with connected undirected graphs, but all results presented here are valid for directed graphs as well. Two vertices v and w are called *adjacent* iff $\{v, w\} \in E$. w is called a *neighbor* of v.

By v and w, we always denote vertices, b and c denote colors, and we assume w.l.o.g. that a vertex-coloring f maps onto $\mathcal{F} := \{1, 2, \dots, n_f\}$, $n_f := |f(V)|$.

Vertices with the same color c are collected in a *color class*

$$\mathcal{C}(c) := \{v \mid f(v) = c\}$$

and we define $\mathcal{C}(v) := \mathcal{C}(f(v))$.

The integers

$$p_v^c := |\{w \in V \mid f(w) = c \text{ and } \{v, w\} \in E\}|$$

are called the *structure values* of G. p_v^c is the number of neighbors of v with color c.

Let

$$L(v) := \{(c, p_v^c) \mid c \in \mathcal{F}, p_v^c \neq 0\}$$

be the *structure set of* v and

$$L(c) := \{(v, p_v^c) \mid v \in V, p_v^c \neq 0\}$$

be the *structure set of* c. In $L(v)$, the numbers of neighbors of the vertex v distinguished by their colors are collected. $L(c)$ collects vertices and their number of neighbors with color c.

A *partition* $\mathcal{P} = \{P_1, P_2, \dots, P_r\}$, $\bigcup_{i=1}^r P_i = V$, of the vertex set is called *equitable* iff

$$v, w \in P_i \Leftrightarrow L(v) = L(w).$$

A vertex-coloring is called *equitable* iff

$$f(v) = f(w) \Leftrightarrow L(v) = L(w).$$

Let $S \subseteq V$ be a set of vertices. We define

$$\overline{N}(S) := \{w \in V \mid \exists v \in S : \{v, w\} \in E\}.$$

We write $\overline{N}(v)$ instead of $\overline{N}(\{v\})$. Observe that this definition differs from the standard definition of the neighborhood of a set of vertices.

The standard algorithm for computing the coarsest equitable coloring can be stated as follows.

Algorithm 1 Equitable Coloring

Input: $G = (V, E, f)$

Output: A equitable coloring f of G

```
1: repeat
2:     compute $L(v)$, $\forall v \in V$
3:     splitcolor, i.e., $\tilde{f}(v) = \tilde{f}(w) :\Leftrightarrow L(v) = L(w)$ and $f(v) = f(w)$, $v, w \in V$
4:     recolor, i.e., $f \equiv \tilde{f}$
5: until $n_f$ did not change
```

One iteration of **Algorithm 1** is called a *step* (lines 2-4) and will be denoted by ⟨step⟩. We refer to $\tilde{f}$ as pseudo coloring of the vertices. $\tilde{f}$ is needed to define the set of *new colors* $\mathcal{N}$. $\mathcal{N}$ is initially defined as the set $f(V)$ and will be recomputed directly prior to every ⟨recolor⟩ operation as $\mathcal{N} := \tilde{f}(V) \setminus f(V)$.

A straightforward analysis yields a worst-case running time of $O(n^3 \log(n))$ since the best known bound on the number of steps is n. Examples with $\frac{n}{2}$ steps can easily be constructed.

Aho et al. [AHU74] have shown that one can avoid to compute all the structure sets in each step. Moreover, they suggest that if a color class is split into several classes, the largest new color class keeps the old color and the other new classes obtain new colors. Using the fact that the structure set of a vertex has to be taken into account only if one of its neighbors obtained a new color during the last recoloring, each vertex has to be recolored at most $\log(n)$ times.

Taking this into account, they are able to proof a worst-case time bound of $O(m \log(n))$. The major disadvantage of this procedure is that it could happen that the sum of the sizes of the computed structure sets in some steps are in $\Omega(m)$ and thus, the splitcolor procedure becomes costly. It would suffice to compute only the structure sets belonging to one color class at a time. But still, the sum of the sizes of these sets could still be in $\Omega(m)$.

3 The new approach

The main idea of our approach is to compute only parts of the structure sets of the vertices but for all vertices at a time. I.e., we compute the sets $L(c)$ instead of $L(v)$. So ⟨step⟩ can be reformulated as follows.

Here, the graph is stored using adjacency lists, i.e., for every vertex, we store a list of its neighbors, and the vertices are stored in an array of size n. This algorithm (**Algorithm 1** with the procedure ⟨step⟩) requires only linear (in m) space. We are now going to prove the time bound of $O(m \log(n))$.

For this, it is necessary to apply the results of Aho et al. to the new algorithm. It will be shown that ⟨compute $L(c)$⟩ can be implemented in $O(|\overline{N}(\mathcal{C}(c))| + |\mathcal{C}(c)|)$ time, ⟨splitcolor⟩ can be implemented in time $O(|\overline{N}(\mathcal{C}(c))|)$, and ⟨recolor⟩ can be implemented in $O(n)$ time.

Procedure 2 ⟨step⟩

```
1: $\tilde{f} \equiv f$
2: for all $c \in \mathcal{N}$ do
3:     compute $L(c)$
4:     splitcolor($c$), i.e., split the colors in the following way:
       $\tilde{f}(v) = \tilde{f}(w) :\Leftrightarrow \tilde{f}(v) = \tilde{f}(w)$ and $p_v^c = p_w^c, \quad \forall v, w \in V.$
5: end for
6: recolor
```

To see this, it is necessary to describe the data structure in more detail. The color classes are stored in an array. Each class consists of a doubly linked list between its members. Furthermore, every vertex knows its color. The structure sets $L(c)$ are stored as doubly linked lists, similar to the color classes. This makes it possible to carry out append, delete and update operations in $O(1)$.

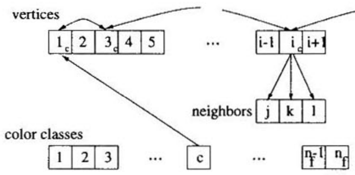


Figure 1: The color class c consists of the elements $1, 3, \ldots, i, \ldots n$, and i has the neighbors j, k, l.

To present a fast version of ⟨splitcolor⟩, we need some more notation. A vertex v is called *hit* by c if $p_v^c > 0$, a color class $\mathcal{C}(b)$ is called *hit* by c if some $v \in \mathcal{C}(b)$ is hit by c. $\mathcal{C}(b).hit$ denotes the number of hit elements of $\mathcal{C}(b)$. This number is needed in ⟨splitcolor⟩ and can easily be computed in **Procedure 3** (which is done in lines 3-5).

Procedure 3 ⟨compute $L(c)$⟩

```
1: for all $w \in \mathcal{C}(c)$ do
2:     for all $v \in \overline{N}(w)$ do
3:         if $p_v^c == 0$ then
4:             $\mathcal{C}(\tilde{f}(v)).hit++$
5:             append $(v, p_v^c)$ to $L(c)$
6:         end if
7:         $p_v^c++$
8:         update the entry of $(v, p_v^c)$ in $L(c)$
9:     end for
10: end for
```

Obviously, the time for computing the structure set $L(c)$, needed in line 3 of **Procedure 2** is bounded by $O(|\overline{N}(\mathcal{C}(c))| + |\mathcal{C}(c)|)$ (see **Procedure 3** for details). Since each vertex is recolored at most $\log(n)$ times, the sum over the computing times of all structure sets computed during the execution of the algorithm is bounded by $O(m\log(n))$.

We now turn to the analysis of **Procedure 4** ⟨splitcolor⟩. Using bucket sort, the sorting of $L(c)$ by the values p_v^c (line 1) can be bounded by $O(|L(c)|)$.

Procedure 4 ⟨splitcolor(c)⟩

```
1: sort $L(c)$ by the values $p_v^c$
2: for all $v$ is first vertex in $L(c)$ with color $\tilde{f}(v)$ do
3:   $b = \tilde{f}(v)$
4:   if $\mathcal{C}(b).hit < \mathcal{C}(b).size$ then
5:     $\mathcal{C}(b).current\_p := 0$
6:   else
7:     $\mathcal{C}(b).current\_p := p_v^c$
8:   end if
9:   $\mathcal{C}(b).current\_color := b$
10:  $\mathcal{C}(b).hit := 0$
11: end for
12: for all $v \in L(c)$ do
13:   if $\mathcal{C}(\tilde{f}(v)).current\_p \neq p_v^c$ then
14:     $\mathcal{C}(\tilde{f}(v)).current\_p := p_v^c$
15:     $\mathcal{C}(\tilde{f}(v)).current\_color := n_{\tilde{f}} + 1$
16:   end if
17:   $\mathcal{C}(\tilde{f}(v)).size - -$
18:   $\tilde{f}(v) := \mathcal{C}(\tilde{f}(v)).current\_color$
19:   $\mathcal{C}(\tilde{f}(v)).size + +$
20: end for
```

In ⟨splitcolor⟩, the pseudo recoloring will be done in the following way. New pseudo colors are assigned according to an increasing ordering of the values p_v^c.

In **Procedure 4**, we determine the smallest p_v^c of each color class $\mathcal{C}(b)$ hit by c (stored in $\mathcal{C}(b).current\_p$). Observe that if some vertices of $\mathcal{C}(b)$ are not hit by c, i.e., $p_v^c = 0$, they do not appear in $L(c)$. It is not possible to find the smallest p_v^c by scanning through all elements of $\mathcal{C}(b)$ because $\mathcal{C}(b)$ or at least the sum of the sizes of all hit color classes might be too large. We want vertices with the smallest p_v^c to keep their old (pseudo) color and the other ones obtain new (pseudo) colors. These temporary (pseudo) colors will be reassigned in ⟨recolor⟩. One possible solution for computing the smallest p_v^c is shown in lines $2-11$. These lines need the sizes of the of the (pseudo) color classes which are updated in lines 17,19. In lines $12-20$, the new pseudo colors are allocated and assigned as described before. Summing up this discussion, **Procedure 4** has an overall running time of $O(|L(c)|) = O(|\overline{N}(\mathcal{C}(c))|)$ and thus, we obtain an overall running time of this procedure of $O(m\log(n))$.

Procedure 5 ⟨recolor⟩

```
1: Let $L$ be the list of all vertices which got a new pseudo color.
2: for all $v \in L$ do
3:   delete $v$ from its color class $\mathcal{C}(f(v))$
4:   append $v$ to $\mathcal{C}(\tilde{f}(v))$
5: end for
6: for all $c = f(v), v \in L$ do
7:   compute d with $|\mathcal{C}(d)| = \max\{|\mathcal{C}(\tilde{f}(v))| \mid f(v) = c\}$
8:   if $|\mathcal{C}(d)| > |\mathcal{C}(c)|$ then
9:     exchange the colors of the color classes $\mathcal{C}(c)$ and $\mathcal{C}(d)$
10:  end if
11: end for
12: update $f$
```

To finish the step, we have to transform the pseudo colors distributed by ⟨splitcolor⟩ into a new coloring. ⟨recolor⟩ (see **Procedure 5**) ensures that the largest color classes keep their old colors and does the updating of the color classes and colors. This is necessary for the validity of the results of Aho et al..

Computing the list L (line 1) can be done by keeping track of the new colors during each ⟨step⟩. In order to update our data structures, the vertices have to be moved from their old color class to their new one. In our data structures, deleting an element from its color class and appending an element to a new class takes time $O(1)$. Thus, lines 2-5 of procedure ⟨recolor⟩ take only $O(n)$ time. Since the sizes of the new color classes are known, all executions of line 7 during one execution of **Procedure 5** take time $O(|L|)$. Hence, lines 6-11 take time $O(|L|)$ as well since two colors will be exchanged iff the new color is larger than the old one. The final line of this procedure can be implemented in time linear in $|L|$.

We conclude that all statements of **Procedure 5** can be executed in time $O(|L|) = O(n)$.

Theorem 1 *The* **Algorithm 1** *together with procedure* **Procedure 2** *has a worst-case running time of* $O(m \log(n))$.

The algorithm presented above does not compute a canonical coloring but the algorithm can be easily adjusted to compute a canonical coloring.

4 The Implementation

The above algorithm is implemented in the program **qstab**[1] which was written in C++.

[1]The program is available at
`http://www-m9.mathematik.tu-muenchen.de/~bastert/qstab.html`.

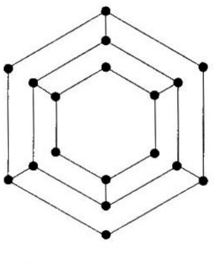
We like to introduce some ideas which decrease the running time of the algorithm considerably. Instead of refining the coloring by considering all necessary colors c and then recoloring once, we refine the coloring by considering one color only and then recolor immediately. This results in much more recolorings and blasts the theoretical time bound but reduces the sizes of the color classes very quickly. This leads to a practically much more efficient algorithm.

One can observe that many structure sets will not yield a refinement of the coloring. It is easy to check whether a color class $\mathcal{C}(b)$ will be split by a structure set $L(c)$ or not. If

$$\max\{p_v^c \mid v \in \mathcal{C}(b)\} \cdot |\mathcal{C}(b)| = \sum_{v \in \mathcal{C}(b)} p_v^c,$$

then the color class $\mathcal{C}(b)$ will not be split, otherwise it will. If we check this condition after computing the structure set and then reduce the structure set if possible, the running time decreases.

To give an idea of the speed of the algorithm, we have tested it on two graph classes, namely benzene stacks and dynkin graphs. Two example instances are depicted in **Figure 2**.


(a) The benzene stack with 18 vertices

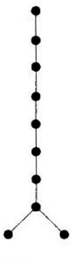

(b) The dynkin graph with 10 vertices

Figure 2: The test instances

Table 1 shows the CPU times in seconds and the number of colors of the coarsest equitable partition of each instance. The results have been obtained on a PC Pentium II/350, Linux 2.2.7, 64 Mb memory, using the GNU C/C++ compiler, version 2.7.2.

Besides the number of vertices, the computation time depends highly on the number of colors of the resulting partition.

	`benzene stack`		`dynkin graph`	
vertices	colors	time	colors	time
200004	33334	**135.19**	200003	**802.56**
150000	25000	**76.27**	149999	**452.39**
100002	16667	**34.25**	100001	**201.28**
10002	1667	**0.4**	10001	**2.1**
1002	167	**0**	1001	**0.02**
102	17	**0**	101	**0**
12	2	**0**	11	**0**

Table 1: Computational results

References

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[Bas98] O. Bastert. New ideas for canonically computing graph algebras. Technical Report TUM-M9803, Technische Universität München, 1998.

[CG97] A. Chan and Ch. D. Godsil. Symmetry and eigenvectors. In G. Hahn and G. Sabidussi, editors, *Graph Symmetry: Algebraic Methods and Applications*, volume 497 of *NATO ASI Series C*, pages 75–106. Kluwer, 1997.

[CRS97] D. Cvetković, P. Rowlinson, and S. Simić. *Eigenspaces of Graphs*. Encyclopedia of Mathematics and Its Applications. Cambridge University Press, 1997.

[God93] Ch. D. Godsil. *Algebraic Combinatorics*. Chapman & Hall, 1993.

[McK81] B. D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

[Sac66] H. Sachs. Über Teiler, Faktoren und charakteristische Polynome von Graphen I. *Wiss. Z.*, 12:7–12, 1966.

[Sac67] H. Sachs. Über Teiler, Faktoren und charakteristische Polynome von Graphen II. *Wiss. Z.*, 13:405–412, 1967.

[Sch74] A. J. Schwenk. Computing the characteristic polynomial of a graph. In R. Bari and Frank Harary, editors, *Graphs and Combinatorics*, pages 153–172. Springer, Berlin, 1974.

[ST99] P. F. Stadler and G. Tinhofer. Equitable partitions, coherent algebras and random walks: Applications to the correlation structure of landscapes. *Match*, this volume, 1999.