Advanced Algorithms (III)

Yijia Chen
Fudan University

Color Refinement

A rough sketch

1. Initially, all vertices have the same colour.
2. Then in each step of the iteration, two vertices that currently have the same colour get different colours if for some colour c they have a different number of neighbours of colour c.
3. The process stops if no further refinement is achieved, resulting in a stable colouring of the graph.

GI by color refinement

1. Run color refinement on the disjoint union of two graphs G and H.
2. If the stable colouring differs on G and H, that is, if for some colour c, the graphs have a different number of vertices of colour c, then they are nonisomorphic.

Stable partitions

$\pi = \{S_1, \ldots, S_k\}$ is a partition of a set V if

- $S_i \neq \emptyset$ for every $i \in [k]$,
- $\bigcup_{i \in [k]} S_i = V$,
- $S_i \cap S_j = \emptyset$ for every $1 \leq i < j \leq k$.

$u \approx_\pi v$ if there exists some $S \in \pi$ with $u, v \in S$.

Let $G = (V, E)$ be a graph. A partition π of V is stable for G if

$$\left|N(u) \cap S\right| = \left|N(v) \cap S\right|$$

for every $u \approx_\pi v$ and $S \in \pi$.

A key lemma

Lemma

If π is stable and $\deg(u) \neq \deg(v)$, then $u \not\approx_\pi v$.

Refinement

A partition ρ of V refines a partition π of a subset S of V if for every $u, v \in S$,

$$u \approx_\rho v \quad \Longrightarrow \quad u \approx_\pi v.$$

$\pi \preceq \rho$ if ρ refines π. If in addition $\rho \neq \pi$, then $\pi \prec \rho$.

Refining operation

A set $V' \subseteq V$ is π-closed if there exist some $S_{i_1}, \ldots, S_{i_\ell} \in \pi$ with

$$V' = \bigcup_{i \in [\ell]} S_{i_j}.$$

Let $G = (V, E)$ be a graph, and let π and π' be partitions of V. For vertex sets $R, S \subseteq V$ that are π-closed, we say that π' is obtained from π by a refining operation (R, S) if

- $S' \in \pi'$ for every $S' \in \pi$ with $S' \cap S = \emptyset$;
- let $u, v \in S$, then $u \approx_{\pi'} v$ if and only if $u \approx_\pi v$ and

$$\left|N(u) \cap R'\right| = \left|N(v) \cap R'\right|$$

  for all $R' \in \pi$ with $R' \subseteq R$.

Refining operation (cont'd)

If π' is obtained from π by a refining operation (R, S), then $\pi \preceq \pi'$.

The operation (R, S) is effective if $\pi \prec \pi'$.

Lemma
An effective refining operation exists for π if and only if π is unstable.

Lemma
Let π' be obtained from π by a refining operation (R, S). If ρ is a stable partition with $\pi \preceq \rho$, then $\pi \preceq \pi' \preceq \rho$.

Lemma
Let $G = (V, E)$ be a graph. For every partition π of V , there is a unique coarsest stable partition ρ that refines π. That is, $\pi \prec \rho$ and for every stable ρ' with $\pi \preceq \rho'$ we have $\rho \preceq \rho'$.

Canonicity

Let $\mathbb{A}$ be an algorithm which on every input G produces a sequence

$$S_1^G, \ldots, S_{k(G)}^G \subseteq V(G).$$

$\mathbb{A}$ is canonical if for every pair of isomorphic graphs G and G' and every isomorphism $h : V(G) \to V(G')$

1. $k(G) = k(G')$,
2. $v \in S_i^G$ implies $h(v) \in S_i^{G'}$ for every $v \in V(G)$ and $i \in [k(G)]$.

Example

For ever $i \leq |V(G)|$ let

$$S_i^G := \{v \mid \deg(v) = i - 1\}.$$

Then

$$S_1^G, \ldots, S_{|V(G)|}^G$$

is canonical.

The algorithm

The algorithm (1)

The algorithm outputs an ordered partition of $V(G)$

$$C_1, \ldots, C_k$$

such that $\{C_1, \ldots, C_k\}$ is the coarsest stable partition of G.

The algorithm (2)

The input is a graph $G = (V, E)$ with $V = [n]$. For every $v \in V$ we are given

$$N(v) := \{u \mid \{u, v\} \in E\},$$

i.e., the neighbor of v in G.

The algorithm maintains an ordered partition $\pi = C_1, \ldots, C_k$ of V, starting with the unit partition.

This partition is iteratively refined using operations of the form (R, V), where $R = C_r$ for some $r \in [k]$.

We will show that when the algorithm terminates, no effective refining operations are possible on the resulting partition. So the resulting partition is the unique coarsest stable partition of G.

The algorithm (3)

Let $\pi = C_1, \ldots, C_k$ be the current partition. Every $i \in [k]$ is a color, and C_i its correspond color class.

S_{refine} is a canonical sequence (stack) of colors that should still be used as refining colours. In particular, we choose a new refining color r to be the last one added to S_{refine}.

Let $R = C_r$, and for any $x \in V$

$$d_r(x) := \left|N(x) \cap R\right|$$

is the color degree of x.

We partition every color class C_s into new classes $C_{\sigma_1}, \ldots, C_{\sigma_p}$ such that for $x \in C_{\sigma_i}$ and $y \in C_{\sigma_j}$:

1. $i = j$ if and only if $d_r(x) = d_r(y)$,
2. if $i < j$ then $d_r(x) < d_r(y)$.

We choose $\sigma_1 = s$, and $\sigma_i = k + i - 1$ for all $2 \leq i \leq p$ and then update the number of colors $k := k + p - 1$.

The algorithm (4)

Initially, S_{refine} contains color 1 with $C_1 = V$. Whenever new colors are introduced during the splitting of a color class C_s, these are pushed onto the stack S_{refine}, in increasing order.

There is one exception: if we have already used C_s as refining color class before, and now C_s is split up to $C_{\sigma_1}, \ldots, C_{\sigma_p}$, then it is not necessary to use all of these new colors as refining colors later.

For every $i \in [p]$ and $x, y \in V(G)$, if

$$\left|N(x) \cap C_s\right| = \left|N(y) \cap C_s\right| \text{ and } \left|N(x) \cap C_{\sigma_j}\right| = \left|N(y) \cap C_{\sigma_j}\right| \text{ for every } j \neq i.$$

then $\left|N(x) \cap C_{\sigma_i}\right| = \left|N(y) \cap C_{\sigma_i}\right|$.

To be precise, for a given s, we canonically choose b to be the minimum colour degree that maximizes $\left|\{x \in C_s \mid d_r(x) = b\}\right|$, and add all newly introduced colors to the stack, in increasing order, except the one that corresponds to b.

The algorithm (5)

The algorithm terminates when S_{refine} is empty, and returns the final ordered partition $C_1, \ldots, C_k$.

The analysis

The correctness

Lemma

For any graph G, the above algorithm computes a canonical sequence $C_1, \ldots, C_k$, such that $\{C_1, \ldots, C_k\}$ is the coarsest stable partition of G.

Proof

$\pi = \{C_1, \ldots, C_k\}$ is refined by the coarsest stable partition ω of G, because it is obtained from the partition $\{V\}$ by using refining operations. It is then equal to ω once we prove that π is stable.

One can verify that the following invariant is maintained:

> *if there exist C_r and C_s such that (C_r, C_s) is effective, then S_{refine} contains a color r' such that $(C_{r'}, C_s)$ is effective.*

Implementation and complexity Bound

We consider one iteration consisting of

1. popping a refining color r from $\S_{\text{refine}}$,
2. applying the refining operation (R, V), with $R = C_r$.

Lemma

The ith iteration takes time

$$O\big(|R| + D(R) + k_i \log k_i\big),$$

where $D(R) = \sum_{v \in R} \deg(v)$ and k_i the new colors introduced in this iteration.

Proof (1)

Computing the colour degrees $d_r(v)$ efficiently for every $v \in V$:

- Use an array cdeg[v] indexed by $v \in [n]$.
- At the beginning of every iteration, cdeg[v] $= 0$ for all v.
- Compute these color degrees by looping over all neighbours w of all $v \in R$, and increasing cdeg[w].
- Compute a list C_{adj} of colors s that contain at least one vertex $w \in C_s$ with cdeg[w] ≥ 1, and for every such color s, we compute a list A_s of all vertices w with cdeg[w] ≥ 1.

This can all be done in time

$$O(|R| + D(R)).$$

Proof (2)

How to consider all colors that split up in one iteration, in canonical (increasing) order:

- Compute a list C_{split}, which represents the subset of C_{adj} containing all colors that actually split up.
- By ensuring that all colors in C_{split} split up, we have that $\left|C_{\text{split}}\right| \leq k_i$, and therefore we can sort this list in time $O(k_i \log k_i)$.
- To compute which colors split up, we compute for every color $s \in C_{\text{adj}}$ the maximum color degree maxcdeg[s] and minimum color degree mincdeg[s].

Proof (3)

How to split a single color class $S = C_s$ and add appropriate new colors to S_{refine}, all in time $O(D_R(S))$ with $D_R(S) = \sum_{v \in S} \left| N(v) \cap R \right|$:

- For every relevant d, we compute how many vertices in C_s have colour degree d, i.e., numcdeg[d].
- Using numcdeg, we can easily compute the (minimum) color degree b that occurs most often in S, which corresponds to the new color that is possibly not added to S_{refine}.
- Construct an array f_{newcol}, indexed by $d \in \{0, \ldots, \text{maxcdeg}[s]\}$, which represents the mapping from color degrees that occur in S to newly introduced colors, or to the current color s.
- Finally, we can loop over A_s in time $O(D_R(S))$, and move all vertices $v \in A_s$ from C_s to C_j, where $j = f_{\text{newcol}}\left[\text{cdeg}[v]\right]$.