# More about CFTP

Application on $k$-colouring

Yanheng Wang

In the lecture, we studied *monotone* CFTP. That is, the state space $\mathcal{X}$ is equipped with a partial order $\leq$, and the transition function $f$ preserves the order.

This could be exponential!

Monotonicity is nice because we don't have to run $|\mathcal{X}|$ chains in parallel; instead, we only run two of them – the *top* chain and the *bottom* chain. If they meet, then we safely assert that all the chains meet (i.e. the space collapse to a singleton).

However, in situations such as $k$-colouring problem, there's no apparent partial order and compatible transition function.

Without the help of monotonicity, we must find some alternative way to tell whether the space collapses. (Running all the chains in parallel is clearly prohibitively expensive.)

```
$T := -1/2$
**repeat**
    $T := 2T$
    **for** $t := T + 1 \ldots 0$ **do**
        Choose i.i.d $S_t \sim S$ if haven't chosen yet
        Generate $f_t := f(\cdot, S_t)$
**until** *the function* $g_T^0$ *collapses the space* $\mathcal{X}$;
**return** the image of $g_T^0$
```

Non-trivial!

For the $k$-colouring problem, an elaborate construction is given by Bhandari and Chakraborty this year. We shall describe their idea in the rest of the slide.

**Main result.** Given a graph $G$ with $k > 3\Delta$, we can construct a perfect uniform sampler that produces a proper $k$-colouring of $G$. Moreover, the sampler runs efficiently (in the sense of expectation).

The construction of the transition function $f$ lies central. $f$ is a complicated function that contains many steps. (It's so complicated that we'd better describe it in the language of algorithm.)

**Notation.** We denote the neighbourhood of $v$ as $\Gamma(v)$.
**Notation.** Denote $\Psi(v) := \{1, 2, \dots, k\} - x(\Gamma(v))$. In other words, $\Psi(v)$ is the set of allowable colours at $v$ in colouring $x$.

```
Function $f(x, s)$
    $L[v] := \{1, 2, \dots, k\}$ for all $v \in V$
    foreach $v \in V$ do
        Select a set of colours $\mathcal{C}$ smartly
        foreach $u \in \Gamma(v)$ do
            $\textsc{Compress}(u, \mathcal{C}, s)$
        $\textsc{Contract}(v, s)$
    Decode a sequence of vertices, $R$, from $s$
    foreach $v \in R$ do
        $\textsc{Contract}(v, s)$
    return $x$
```

They will use or update $x$ and $L$!

# The Rationale Behind $f$

The array $L$ is an important tracker that changes as the program executes. $L[v]$ aggregates *all* possible values of $x(v)$ *at the moment*, when input $x$ ranges over $\mathcal{X}$.

```
Function f(x, s)
    $L[v] := \{1, 2, \ldots, k\}$ for all $v \in V$
    foreach $v \in V$ do
        Select a set of colours $\mathcal{C}$ smartly
        foreach $u \in \Gamma(v)$ do
            COMPRESS$(u, \mathcal{C}, s)$
        CONTRACT$(v, s)$
    Decode a sequence of vertices, $R$, from $s$
    foreach $v \in R$ do
        CONTRACT$(v, s)$
    return $x$
```

It loops over $v \in V$ one by one, calls CONTRACT to modify $x(v)$ geniously so that $|L[v]|$ is contracted down to 2. However, before we call it, we must spruce up $\Gamma(v) = \{u_1, \ldots, u_p\}$ using the inner loop. COMPRESS overlaps $L[u_1], \ldots, L[u_p]$ so that they share a core $\mathcal{C}$ of size $\Delta$. (Similar to a sunflower in combinatorics.) Specifically, $|L[u_i] - \mathcal{C}| \leq 1$.

The final part continues modifying $x$ so that $|L[v]| = 1$ for all $v$ with high probability. Note that if we end up $\forall v : |L[v]| = 1$, then we are safe to say that $f(\cdot, s)$ collapses the space $\mathcal{X}$ to a singleton!

Let $f$ and $S$ induce $P$. They are designed to make $\mu P = \mu$ for uniform distribution $\mu$.

Since the random source $s$ encodes tons of information we need (including those used in subroutines), it's tedious to write it formally.

We would rather adopt the casual style: When we discuss the subroutines, we use descriptions such as "generate a random vertex" and "select a random colour". You should keep in mind (and verify) that *such randomness are in fact provided by $s$, and shall not depend on $x$.*

# Subroutine Compress

Recall that our goal here is to "concentrate" $L[u]$: We shall make $L[u] = \mathcal{C} \cup \{c\}$ when Compress finishes. In other words, at the moment Compress finishes, $x(u)$ *always* take values in $\mathcal{C} \cup \{c\}$, *regardless of what the input $x$ looks like.*

```
Shuffle $\mathcal{C}$ randomly
Select a random colour $c \notin \mathcal{C}$
Pick at random $\tau \in [0, 1]$
**if** $c \in \Psi(u)$ *and* $\tau \le p := \frac{k-\Delta}{|\Psi(u)|}$ **then**
    $x(u) := c$
**else**
    $x(u) :=$ the first feasible colour in $\mathcal{C}$
$L[u] := \mathcal{C} \cup \{c\}$
```

The extra colour compensates for the absence of $\Psi(u) - \mathcal{C}$. It's essential to mimic the Glauber dynamics.

It always succeeds, assuming $|\mathcal{C}| = \Delta$. If $c \notin \Psi(u)$ then $\exists w \in \Gamma(u)$ such that $x(w) \notin \mathcal{C}$, thus at most $\Delta - 1$ vertices in $\Gamma(u)$ use the colours in $\mathcal{C}$. If $\tau > p$ then $k - \Delta < |\Psi(u)|$ since $\tau \le 1$. This implies $|x(\Gamma(u))| < \Delta$.

When Compress finishes:

- $x(u) \in L[u] = \mathcal{C} \cup \{c\}$ indeed, regardless of the input $x$.
- It implements Glauber dynamics at $u$. (See next page.)

Q: *Why should we care about Glauber dynamics here?*

A: In a word, it helps us establish $\mu P = \mu$. If we can show that Compress implements Glauber dynamics at vertex $u$, then the uniform distribution $\mu$ must be a *fixed point* of Compress. (**Exercise: Figure out why.**)

Now we show that Compress implements Glauber dynamics at $u$. For each colour $c_0 \in \Psi(u)$,

| | |
|---|---|
| $\underline{c_0 \notin \mathcal{C}}$ | $\Pr[x(u) = c_0] = p \cdot \frac{1}{k - \lvert\mathcal{C}\rvert} = \frac{1}{\lvert\Psi(u)\rvert}$ |
| $\underline{c_0 \in \mathcal{C}}$ | Notice that $\sum_{i \in \Psi(u)} \Pr[x(u) = i] = 1$ since we never produce illegal colourings. Excluding the probability of previous case, we have total mass $\frac{\lvert\Psi(u) \cap \mathcal{C}\rvert}{\lvert\Psi(u)\rvert}$ on space $\Psi(u) \cap \mathcal{C}$. But since we chose the first feasible colour from $\mathcal{C}$, which was shuffled randomly, the mass must be uniformly distributed on $\Psi(u) \cap \mathcal{C}$. Therefore, $\Pr[x(u) = c_0] = \frac{1}{\lvert\Psi(u)\rvert}$. ■ |

## Subroutine Contract

Our goal here is even more radical: Contracting $|L[v]|$ to 2. Furthermore, with some good chance we could actually decrease the size to 1!

**More notations.**

$$B[v] := \bigcup_{u \in \Gamma(v)} L[u] \qquad \Bigg| \qquad A[v] := \bigcup_{\substack{u \in \Gamma(v) \\ |L[u]|=1}} L[u]$$

These two quantities vary as $f$ executes. At any instant, $A[v] \subseteq x(\Gamma(v)) \subseteq B[v]$. (**Exercise: Prove it.**)

**Remark.** For Contract to be effective, we must ensure that $|B[v]| \leq k - \Delta$ in advance. That's why we must spruce the neighbourhood up before we apply Contract. After the cleansing, we have $|L[u] - \mathcal{C}| \leq 1$ for all $u \in \Gamma(v)$, hence $|B[v]| \leq |\mathcal{C}| + \Delta = 2\Delta < k - \Delta$.

```
Select a random colour $c_1 \in \{1, 2, \ldots, k\} - B[v]$
Select a random colour $c_2 \in B[v] - A[v]$
Pick at random $\tau \in [0, 1]$
if $c_2 \in \Psi(v)$ and $\tau \leq p := \frac{|B[v]-A[v]|}{|\Psi(v)|}$ then
    $x(v) := c_2$
else
    $x(v) := c_1$
if $\tau \leq p' := \frac{|B[v]-A[v]|}{k-\Delta}$ then
    $L[v] := \{c_1, c_2\}$
else
    $L[v] := \{c_1\}$
```

Always a safe choice

Compensate for vacuum area

This part looks weired. Why don't we update $L$ above? Our purpose is to *separate* $L$ and $x$ – the update of $L$ never depends on the specific value of $x$. (See the final page for reason.) Beware that $\Psi$ does depend on $x$, thus we could only *estimate* the threshold without the knowledge of $x$.

By similar analysis to the subroutine COMPRESS, we establish the following properties when CONTRACT finishes.

- $x(v) \in L[v]$ indeed, regardless of the input $x$.
- It implements Glauber dynamics at $v$.

# The Art of Choosing $\mathcal{C}$

```
Function f(x, s)
    L[v] := {1, 2, ..., k} for all v ∈ V
    foreach v ∈ V do
        Select a set of colours C smartly
        foreach u ∈ Γ(v) do
            Compress(u, C, s)
        Contract(v, s)
    Decode a sequence of vertices, R, from s
    foreach v ∈ R do
        Contract(v, s)
    return x
```

```
Function f(x, s)
    L[v] := {1, 2, ..., k} for all v ∈ V
    foreach v ∈ V do
        Select a set of colours C smartly
        foreach u ∈ Γ(v) and u > v do
            Compress(u, C, s)
        Contract(v, s)
    Decode a sequence of vertices, R, from s
    foreach v ∈ R do
        Contract(v, s)
    return x
```

At this point, we revisit the function $f$. There's a subtlety that we purposely ignored. The goal of the outer loop is to contract $|L[v]|$ down to 2 for *all* $v$, and the prerequisite is to spruce up the neighbourhood of $v$. However, we may unconsciously destroy the previous work (i.e. bringing $|L[v']|$ back to $\Delta + 1$) during the spruce up!

Here's a quick fix: If the vertex $u$ has been contracted before (i.e. $u < v$), then we simply skip it.

But the story isn't over. So long as we skip it, how can we ensure that $|L[u] - \mathcal{C}| \leq 1$?

It turns out that we can accomplish this by choosing a $\mathcal{C}$ that intersects with $L[u]$ for all $u \in \Gamma(v) \wedge u < v$. Expressed in algorithm:

```
$\mathcal{C} := \emptyset$
**foreach** $u \in \Gamma(v)$ *and* $u < v$ **do**
    Pick arbitrary $a \in L[u]$
    $\mathcal{C} := \mathcal{C} \cup \{a\}$
```

Such $u$ must have been contracted in previous rounds, thus $|L[u]| \leq 2$.

Obviously, $|\mathcal{C}| \leq \Delta$. If the size is smaller than $\Delta$, then we append some arbitrary colours to make it $\Delta$.

## Coup de Grâce

Now comes the exciting part. We spot an unusual property: The tracker array $L$ in function $f$ is driven by randomness $s$ only! In other words, even if we don't know the exact input $x$, we could still run $f$ virtually and obtain the correct value of $L$. (Recall the definition of $L$ is independent of specific $x$.)

This is exactly what we want: We could infer the collapse of space *without* running $|\mathcal{X}|$ chains in parallel; instead, we just run $f$ virtually (on arbitrary input $x$) and see what's the value $L$. If we are lucky that $|L[v]| = 1$ for all $v$, then we know that the space indeed collapsed!

## Appendix: Running Time

We supplement the running time analysis here. In $f$, we take $R$ to be a random sequence of vertices of length $2\frac{k-\Delta}{k-3\Delta}n\ln n$ (allowing repetitions). The meaning of this number will be clear soon.

**Theorem.** By the time $f$ returns, $\Pr[\forall v : |L[v]| = 1] \geq 1/2$.

**Corollary.** The algorithm is only expected to call $f$ twice, before it finds the space is collapsed. But we know that each run of $f$ takes polynomial time only, thus the algorithm is expected to terminate in polynomial time.

**Remark.** Intuitively, the theorem says that if we repeat CONTRACT on random vertices for many times, then with high probability we are able to "drift" $|L[v]|$ from 2 to 1 for all $v$.

The proof of the theorem relies on the following lemma, whose proof is omitted here.

**Lemma.** Consider an arbitrary random walk $\{I_t\}$ on $\{0, 1, \ldots, n\}$ satisfying $\forall t : |I_{t+1} - I_t| \leq 1$ with absorbing state $n$. If $\mathbb{E}[I_{t+1} - I_t \mid I_t = i] \geq \kappa(i) > 0$ for all $t$, then

$$\sum_{i=0}^{n-1} \mathbb{E}[H_i] \leq \sum_{i=0}^{n-1} \frac{1}{\kappa(i)}$$

where $H_i :=$ the number of times the walk hits state $i$.

**Idea.** In our context, define $\overset{\text{“Good”}}{\boxed{G}} := \{v \in V \mid |L[v]| = 1\}$ that changes all the time as $f$ proceeds. For clarity, we put a subscript and write it $G_t$. Denote $I_t := |G_t|$. Clearly, $\{I_t\}$ is a random walk on $\{0, 1, \ldots, n\}$, and $n$ is the absorbing state (**Exercise**). Our target is to drift $I_t$ to $n$. We shall use the above lemma to show that we could reach our goal efficiently.

**Lemma.** Consider an arbitrary random walk $\{I_t\}$ on $\{0, 1, \dots, n\}$ satisfying $\forall t : |I_{t+1} - I_t| \leq 1$ with absorbing state $n$. If $\mathbb{E}[I_{t+1} - I_t \mid I_t = i] \geq \kappa(i) > 0$ for all $t$, then

$$\sum_{i=0}^{n-1} \mathbb{E}[H_i] \leq \sum_{i=0}^{n-1} \frac{1}{\kappa(i)}$$

where $H_i :=$ the number of times the walk hits state $i$.

## Proof of the theorem.

Probability that we lose a member in $G_{t+1}$.

$$\mathbb{E}[I_{t+1} - I_t \mid I_t = i] = \boxed{\sum_{v \notin G_t} \frac{1}{n}\left(1 - \frac{|B[v] - A[v]|}{k - \Delta}\right)} - \boxed{\sum_{v \in G_t} \frac{1}{n} \frac{|B[v] - A[v]|}{k - \Delta}}$$

Probability that we have a new member in $G_{t+1}$.

$$= \frac{1}{n}\left((n-i) - \sum_{v \in V} \frac{|B[v] - A[v]|}{k - \Delta}\right)$$

$$\geq \frac{1}{n}\left((n-i) - \frac{2(n-i)\Delta}{k - \Delta}\right)$$

$$= \frac{n-i}{n} \cdot \frac{k - 3\Delta}{k - \Delta} =: \kappa(i)$$

Note $|B[v] - A[v]| \leq 2|\Gamma(v) \cap \overline{G_t}|$ for all $v$. Hence, $\sum |B[v] - A[v]| \leq 2(n-i)\Delta$ since every $w \in \overline{G_t}$ is counted for at most $\Delta$ times in the summation.

Thus $\sum_{i=0}^{n-1} \mathbb{E}[H_i] \leq \sum_{i=0}^{n-1} \frac{1}{\kappa(i)} \leq \frac{k-\Delta}{k-3\Delta} n \ln n$.

**Lemma.** Consider an arbitrary random walk $\{I_t\}$ on $\{0, 1, \ldots, n\}$ satisfying $\forall t : |I_{t+1} - I_t| \leq 1$ with absorbing state $n$. If $\mathbb{E}[I_{t+1} - I_t \mid I_t = i] \geq \kappa(i) > 0$ for all $t$, then

$$\sum_{i=0}^{n-1} \mathbb{E}[H_i] \leq \sum_{i=0}^{n-1} \frac{1}{\kappa(i)}$$

where $H_i :=$ the number of times the walk hits state $i$.

Recall that *our goal is hitting the number $n$.* Put it in another way, we don't want $H_n = 0$. Well, how often could $H_n = 0$ happen? It turns out that $H_n = 0 \iff \sum_{i=0}^{n-1} H_i \geq |R|$, where $|R|$ is the length of our random sequence. Hence,

$$\begin{aligned}\Pr[H_n = 0] &= \Pr\left[\sum_{i=0}^{n-1} H_i \geq |R|\right] \\ &\leq \frac{\mathbb{E}\left[\sum_{i=0}^{n-1} H_i\right]}{|R|} \leq \frac{1}{2}\end{aligned}$$

by Markov inequality. ■