Algorithms and Bioinformatics

Part II — Comparative Genomics

II.3 — More on FPT Algorithms
(some of them in Bioinformatics)

Laurent Bulteau
Dynamic Programming

- Not specific to FPT, but often used in this context
- aka. “table-filling”
- Enumerate polynomialy many subproblems, solve each one by combining results from other (sub-)subproblems
- Other point of view: write a simple recursive program, use a cache to store and re-use intermediate results
Dynamic Programming

**Maximum Agreement Subtree**

**Input:** Two trees $T_1$, $T_2$, with leaf labels

**Output:** Subtrees $T'_1$ of $T_1$ and $T'_2$ of $T_2$, with max. number of leaves, such that $T'_1 = T'_2$ up to degree-2 vertex contraction.

**D.P. table:**

$\text{MAS}(u, v) = \text{size of Maximum Agreement Subtree of } T_1[u], T_2[v]$
Dynamic Programming

Maximum Agreement Subtree

Input: Two trees $T_1$, $T_2$, with leaf labels
Output: Subtrees $T'_1$ of $T_1$ and $T'_2$ of $T_2$, with max. number of leaves, such that $T'_1 = T'_2$ up to degree-2 vertex contraction.

▷ D.P. table: $MAS(u, v) =$ size of Maximum Agreement Subtree of $T_1[u]$, $T_2[v]$
Dynamic Programming

**Maximum Agreement Subtree**

| **Input:** | Two trees $T_1, T_2$, with leaf labels |
| **Output:** | Subtrees $T'_1$ of $T_1$ and $T'_2$ of $T_2$, with max. number of leaves, such that $T'_1 = T'_2$ up to degree-2 vertex contraction. |

- D.P. table: $MAS(u, v) = \text{size of Maximum Agreement Subtree of } T_1[u], T_2[v]$

Color Coding

- General use: find size-\(k\) subsets with specific properties in a large set of elements
- Randomized technique, can be de-randomized
- Best-known use case: find a length-\(k\) simple path in a graph
Minimum Weight Path

**Input:** A (directed) graph $G = (V, E)$, edge weights $w : E \to \mathbb{N}$, integer $k$

**Param.:** $k$

**Output:** A length-$k$ simple path of $G$ with maximum weight

- NP-hard...
- Motivation: find *signaling pathways* in protein-protein interaction networks
Color Coding

PPI Network

A Protein-Protein Interaction Network. \(^1\)

\(^1\) credits: Fan et al., Nature Scientific Reports 8:351, 2018
Extra knowledge can help: what if you know how to split the graph into $k$ classes (*colors*), and know that a solution must use exactly one vertex in each class?
Color Coding
Principle

- Extra knowledge can help: what if you know how to split the graph into $k$ classes (colors), and know that a solution must use exactly one vertex in each class?
- Plus: you know in which order the colors are visited!
Extra knowledge can help: what if you know how to split the graph into $k$ classes (colors), and know that a solution must use exactly one vertex in each class?

Plus: you know in which order the colors are visited!

Exactly how we get this information we shall see later: for now just assume we know this.
Color Coding

With color order
Color Coding
With color order

order:  black  blue  red  yellow
Color Coding
With color order

order:  ●   ○   ▲   ▼
Color Coding
With color order
Color Coding

With color order
Dynamic Programming with color order

Dynamic programming table

For each $u \in V$, what is the maximum weight of a color-consistent path up to $u$? \( \rightarrow W[u] \) (n entries)

Filling the table

If $u$ has color-rank $i$,

$$W[u] = \max_{v \text{ of rank } i-1} (W[v] + w(v \rightarrow u))$$

Border cases:

$$W[u] = 0 \text{ if } u \text{ has rank } 0$$
**Color Coding**

Dynamic Programming without color order

<table>
<thead>
<tr>
<th>Dynamic programming table</th>
</tr>
</thead>
<tbody>
<tr>
<td>For each $u \in V$, and each subset $X \subseteq [k]$ of colors, what is the maximum weight of a path ending in $u$ using once each color in $X$? $\rightarrow W'[u, X]$ ($2^k n$ entries)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Filling the table</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W'[u, X] = \max_v W'[v, X\setminus] + w(v \rightarrow u)$</td>
</tr>
<tr>
<td>Border cases:</td>
</tr>
<tr>
<td>$W'[u, X] = 0$ if $X = {\text{col}(u)}$</td>
</tr>
<tr>
<td>Running time: $O(2^k n^2)$</td>
</tr>
</tbody>
</table>
Color Coding
Without color order
Color Coding
Without color order
Color Coding
Without color order
Color Coding
Without color order
Color Coding

Without color order
Color Coding

How do we pick colors?

Randomly!

Number of colorings of $k$ vertices with $k$ colors: $k^k$

Number of colorful colorings of $k$ vertices with $k$ colors: $k!$

Probability to be colorful on the solution: $\frac{k!}{k^k} \approx e^{-k}$

Number of tries to get constant probability: $\approx e^k$

Key point: this value does not depend on $n$

Randomized FPT algorithm

• Draw $C^e_k$ random colorings of the graph.
• For each one, run the dynamic programming algorithm.

⇒ Running time $O(e^k^2 k n^2)$ (or $O(k^k n^2)$).
Color Coding

How do we pick colors?

- Randomly!
Color Coding
How do we pick colors?

- Randomly!
- Number of colorings of $k$ vertices with $k$ colors: $k^k$
Color Coding
How do we pick colors?

- Randomly!
- Number of colorings of $k$ vertices with $k$ colors: $k^k$
- Number of \textit{colorful} colorings of $k$ vertices with $k$ colors: $k!$
Color Coding
How do we pick colors?

- Randomly!
- Number of colorings of \( k \) vertices with \( k \) colors: \( k^k \)
- Number of \textit{colorful} colorings of \( k \) vertices with \( k \) colors: \( k! \)
- Probability to be colorful on the solution: \( \frac{k!}{k^k} \sim e^{-k} \)
Color Coding
How do we pick colors?

- Randomly!
- Number of colorings of $k$ vertices with $k$ colors: $k^k$
- Number of colorful colorings of $k$ vertices with $k$ colors: $k!$
- Probability to be colorful on the solution: $\frac{k!}{k^k} \sim e^{-k}$
- Number of tries to get constant probability: $\sim e^k$
Color Coding

How do we pick colors?

- Randomly!
- Number of colorings of $k$ vertices with $k$ colors: $k^k$
- Number of colorful colorings of $k$ vertices with $k$ colors: $k!$
- Probability to be colorful on the solution: $\frac{k!}{k^k} \sim e^{-k}$
- Number of tries to get constant probability: $\sim e^k$
- Key point: this value does not depend on $n$
**Color Coding**

How do we pick colors?

- Randomly!
- Number of colorings of $k$ vertices with $k$ colors: $k^k$
- Number of colorful colorings of $k$ vertices with $k$ colors: $k!$
- Probability to be colorful on the solution: $\frac{k!}{k^k} \simeq e^{-k}$
- Number of tries to get constant probability: $\simeq e^k$
- Key point: this value does not depend on $n$

**Randomized FPT algorithm**

- Draw $C.e^k$ random colorings of the graph.
- For each one, run the dynamic programming algorithm.

$\Rightarrow$ Running time $O(e^k2^k n^2)$. 
Color Coding

How do we pick colors?

- Randomly!
- Number of colorings of $k$ vertices with $k$ colors: $k^k$
- Number of colorful colorings of $k$ vertices with $k$ colors: $k!$
- Probability to be colorful on the solution: $\frac{k!}{k^k} \sim e^{-k}$
- Number of tries to get constant probability: $\sim e^k$
- Key point: this value does not depend on $n$

Randomized FPT algorithm

- Draw $C.e^k$ random colorings of the graph. (enumerate $k!$ relative orders on the colors)
- For each one, run the dynamic programming algorithm.

$\Rightarrow$ Running time $O(e^k 2^k n^2)$. 
Color Coding
How do we pick colors?

▶ Randomly!
▶ Number of colorings of $k$ vertices with $k$ colors: $k^k$
▶ Number of colorful colorings of $k$ vertices with $k$ colors: $k!$
▶ Probability to be colorful on the solution: $\frac{k!}{k^k} \approx e^{-k}$
▶ Number of tries to get constant probability: $\approx e^k$
▶ Key point: this value does not depend on $n$

Randomized FPT algorithm

• Draw $C\cdot e^k$ random colorings of the graph. (enumerate $k!$ relative orders on the colors)
• For each one, run the dynamic programming algorithm.

⇒ Running time $O(e^k2^k n^2)$ (or $O(k^k n))$. 
Color Coding

How do we pick colors deterministically?

- **Smart** enumeration of some colorings:
Color Coding
How do we pick colors deterministically?

- **Smart** enumeration of some colorings:
  - Any size-$k$ subset of $V$ is separated at least once into $k$ colors.
Color Coding
How do we pick colors deterministically?

- **Smart** enumeration of some colorings:
  - Any size-\(k\) subset of \(V\) is separated at least once into \(k\) colors.
  - \(k^n\) colorings is too many.
Color Coding
How do we pick colors deterministically?

- **Smart** enumeration of some colorings:
  - Any size-\(k\) subset of \(V\) is separated at least once into \(k\) colors.
  - \(k^n\) colorings is too many.
- Such a list of colorings is called *perfect family of hash functions*
Color Coding

How do we pick colors deterministically?

- **Smart** enumeration of some colorings:
  - Any size-$k$ subset of $V$ is separated at least once into $k$ colors.
  - $k^n$ colorings is too many.

- Such a list of colorings is called *perfect family of hash functions*

---

**Theorem**

There exists a perfect family of hash functions of size $2^{O(k) \log(n)}$ computable in time $2^{O(k)} n \log(n)$. 
Color Coding

How do we pick colors deterministically?

- **Smart** enumeration of some colorings:
  - Any size-$k$ subset of $V$ is separated at least once into $k$ colors.
  - $k^n$ colorings is too many.

- Such a list of colorings is called *perfect family of hash functions*

**Theorem**

There exists a perfect family of hash functions of size $2^{O(k) \log(n)}$ computable in time $2^{O(k) n \log(n)}$.

**Cons:** Additional factor $2^{O(k) \log(n)}$ in the running time
Color Coding

How do we pick colors deterministically?

- **Smart** enumeration of some colorings:
  - Any size-$k$ subset of $V$ is separated at least once into $k$ colors.
  - $k^n$ colorings is too many.

- Such a list of colorings is called *perfect family of hash functions*

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>There exists a perfect family of hash functions of size $2^{O(k) \log(n)}$ computable in time $2^{O(k) n \log(n)}$.</td>
</tr>
</tbody>
</table>

**Cons:** Additional factor $2^{O(k) \log(n)}$ in the running time

**Pros:** Deterministic
Color Coding
How do we pick colors deterministically?

- **Smart** enumeration of some colorings:
  - Any size-\(k\) subset of \(V\) is separated at least once into \(k\) colors.
  - \(k^n\) colorings is too many.
- Such a list of colorings is called *perfect family of hash functions*

**Theorem**

There exists a perfect family of hash functions of size \(2^{O(k) \log(n)}\) computable in time \(2^{O(k) n \log(n)}\).

**Cons:** Additional factor \(2^{O(k) \log(n)}\) in the running time

**Pros:** Deterministic

**Anyway:** Two algorithms in one; let the user choose.
Practice

13. Give an FPT algorithm based on color-coding for the problem below. Bonus: show that it is NP-complete.

### Cheap Subtree

**Input:** A complete binary tree $T$ with a set $L$ of leaves, a graph $G = (V, E)$, a cost function $c : V \times L \to \mathbb{N}$  

**Param.:** $k = |L|$

**Output:** A subset $V' \subseteq V$ such that:
- $G[V']$ is isomorphic to $T$,
- the total cost of the mapping between $V$ and $L$ is minimal.

14. Same question:

### Polychrome Matching

**Input:** A graph $G$ with an $r$-edge coloring  

**Param.:** $r$

**Output:** A maximum-size set of independent edges of $G$ with pairwise-distinct colors.
15. Same question:

**Disjoint r-Subsets**

**Input:** Size-\(r\) subsets \(X_1, \ldots X_m\) of \([n]\), integer \(k\)

**Param.:** \(k + r\)

**Output:** \(k\) pairwise disjoint subsets \(X_{i_1}, \ldots X_{i_k}\)
Color Coding

Final remarks

- Color coding cannot help W[1]-hard problems.

**Multi-Color Clique**

**Input:** A $k$-partite graph $G = (V, E)$, with $V = V_1 \cup \cdots \cup V_k$

**Param.:** $k$

**Output:** A size-$k$ clique $K$, such that $|K \cap V_i| = 1$ for all $i$. 
Color Coding

Final remarks

- Color coding cannot help W[1]-hard problems.

**Multi-Color Clique**

**Input:** A $k$-partite graph $G = (V, E)$, with $V = V_1 \cup \cdots \cup V_k$

**Param.:** $k$

**Output:** A size-$k$ clique $K$, such that $|K \cap V_i| = 1$ for all $i$.

- Use more colors in randomized algorithms: optimal close to $1.3k$ for Minimum Weight Path (fewer trials, but longer dynamic programming)
Iterative Compression

Principle

- Other “heavy” approach, mostly for graphs
- Idea:
  - Start with an empty graph and an empty solution
  - Add vertices (or edges) one by one
  - Each time: update the solution
  - If the solution is too large: compress it by one
- Core algorithm: Given a graph, a target solution size of $k$, and a solution of size $k + 1$, find a solution of size $k$ (if any).
Iterative Compression

**Vertex Cover**

- Start with empty graph, empty solution ($X$)
- Add vertex $v$ (and connecting edges) to $G$ and to $X$
- If $|X| = k + 1$:
  - Partition $X$ into $K$ ("Keep") and $D$ ("Discard")
  - Create $X' = K \cup N(D)$. If $|X'| \leq k$, continue with next vertex.
  - Try with every $2^{k+1}$ branches: reject if no good $X'$.
- Total running time: $O(2^k n^2)$
Iterative Compression

**Odd Cycle Transversal**

**Odd Cycle Transversal**

- **Input:** A graph $G = (V, E)$, an integer $k$
- **Param.:** $k$
- **Output:** A subset $X$ of $G$ such that $G[V \setminus X]$ is bipartite.

- Start with empty graph, empty solution ($X$)
- Add vertex $v$ (and connecting edges) to $G$ and to $X$
- If $|X| = k + 1$:
  - Partition $X$ into 3 parts ($K, L, R$)
  - Create $X'$ using Min-Cut. If $|X'| \leq k$, continue with next vertex.
  - Try with every $3^{k+1}$ branches: reject if no good $X'$.