Path Color Switching

Supervisor: Jean-Charles Régin

Fissore Davide

Mars 30, 2023

MASTER INFORMATIQUE

UNIVERSITÉ CÔTE D'AZUR

Problem Description

We want to generate sequences of musical "chords" with some known constraints as well as control on the complexity of the sequence.

Spotify

Problem Description

> We want to generate sequences of musical "chords" with some known constraints as well as control on the complexity of the sequence.
>
> Spotify

Input An oriented graph whose arcs are colored with a set of colors, two nodes of the graphs s and t and a length k.

Output Set single colors to edges to find a path of length k from s to t minimizing the number of color switches.

Definitions & notations

Color switch (CS): given two adjacent arcs a_1 and a_2 colored respectively with c_1 and c_2, we have a color *CS* if $c_1 \neq c_2$.

Definitions & notations

Color switch (CS): given two adjacent arcs a_1 and a_2 colored respectively with c_1 and c_2, we have a color *CS* if $c_1 \neq c_2$.

$\mathcal{G} = (V, A)$: A directed graph where V is the set of its nodes and A is the set of its arcs.

$\mathcal{C}$: A finite set of colors.

$\mathcal{F}$: The coloring function defined as $\mathcal{F} : A \rightarrow 2^{\mathcal{C}}$.

$\mathcal{P} = (v_1, \ldots, v_k)$: A path going from v_1 to v_k.

Definitions & notations

Color switch (CS): given two adjacent arcs a_1 and a_2 colored respectively with c_1 and c_2, we have a color *CS* if $c_1 \neq c_2$.

$\mathcal{G} = (V, A)$: A directed graph where V is the set of its nodes and A is the set of its arcs.

$\mathcal{C}$: A finite set of colors.

$\mathcal{F}$: The coloring function defined as $\mathcal{F} : A \rightarrow 2^{\mathcal{C}}$.

$\mathcal{P} = (v_1, \ldots, v_k)$: A path going from v_1 to v_k.

$w(\mathcal{P})$: The cost of the path $\mathcal{P}$ which is given by the sum of its *CS*.

Problem decomposition

The problem can decomposed in small parts:

- Minimize *CS* on paths;
- Minimize *CS* on graphs.

Minimize *CS* on Paths

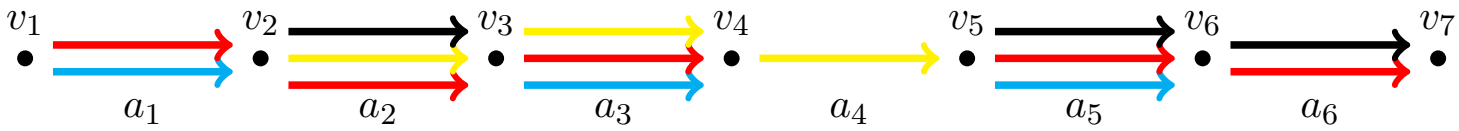


Figure: A path $\mathcal{P}$

What is the color assignation minimizing $w(\mathcal{P})$?

Algorithm

Let $\mathcal{P} = (a_1, \ldots, a_k)$ a path
Let $\mathcal{T} : A \to 2^{\mathcal{C}}$ a function such that:

- $\mathcal{T}(a_1) = \mathcal{F}(a_1)$
- $\mathcal{T}(a_i) = \mathcal{F}(a_i) \cap \mathcal{T}(a_{i-1})$ `if not empty else` $\mathcal{F}(a_i)$

Algorithm

Let $\mathcal{P} = (a_1, \ldots, a_k)$ a path
Let $\mathcal{T} : A \to 2^{\mathcal{C}}$ a function such that:

- $\mathcal{T}(a_1) = \mathcal{F}(a_1)$
- $\mathcal{T}(a_i) = \mathcal{F}(a_i) \cap \mathcal{T}(a_{i-1})$ `if not empty else` $\mathcal{F}(a_i)$

$\mathcal{H} : A \to \mathcal{C}$ the function minimizing $w(\mathcal{P})$ such that:

- $\mathcal{H}(a_k) =$ `a rnd elt from` $\mathcal{T}(a_k)$
- $\mathcal{H}(a_i) = \mathcal{H}(a_{i+1})$ `if it is in` $\mathcal{T}(a_i)$ `else` $\mathcal{T}(a_i)$`.peek()`

Example run

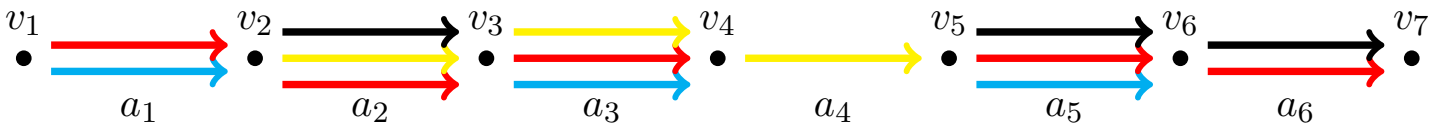


Figure: A path $\mathcal{P}$

Start to compute $\mathcal{T}(\mathcal{P})$

Example run

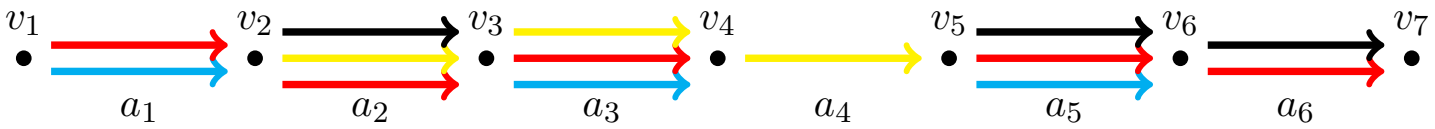


Figure: Computing $\mathcal{T}(\mathcal{P})$

$$\mathcal{T}(a_1) = \mathcal{F}(a_1)$$

Example run

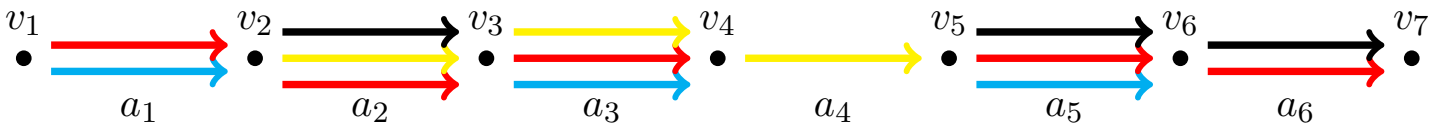


Figure: Computing $\mathcal{T}(\mathcal{P})$

$$\mathcal{T}(a_2) = \mathcal{F}(a_2) \cap \mathcal{T}(a_1) \texttt{ since not empty}$$

Example run

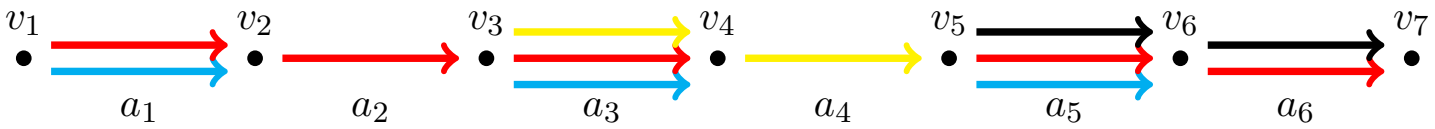


Figure: Computing $\mathcal{T}(\mathcal{P})$

$$\mathcal{T}(a_2) = \mathcal{F}(a_2) \cap \mathcal{T}(a_1) \texttt{ since not empty}$$

Example run

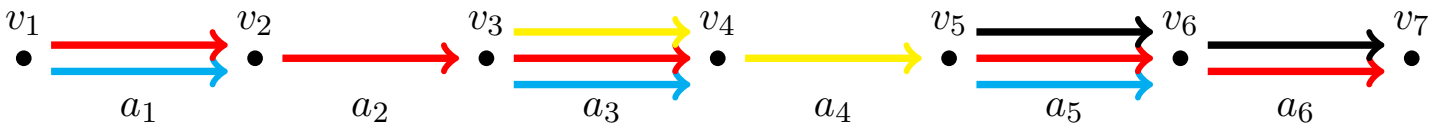


Figure: Computing $\mathcal{T}(\mathcal{P})$

$$\mathcal{T}(a_3) = \mathcal{F}(a_3) \cap \mathcal{T}(a_2) \texttt{ since not empty}$$

Example run

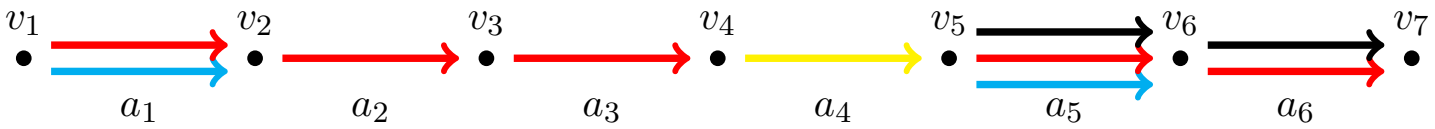


Figure: Computing $\mathcal{T}(\mathcal{P})$

$$\mathcal{T}(a_3) = \mathcal{F}(a_3) \cap \mathcal{T}(a_2) \texttt{ since not empty}$$

Example run

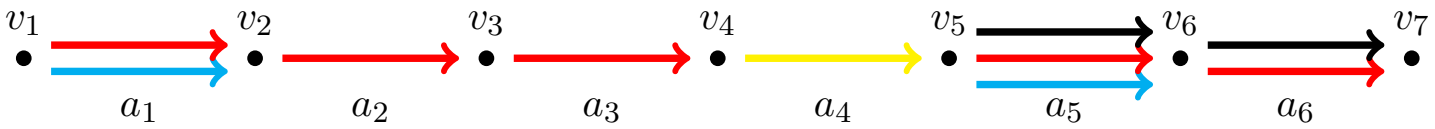


Figure: Computing $\mathcal{T}(\mathcal{P})$

$$\mathcal{T}(a_4) = \mathcal{F}(a_4) \texttt{ since } \mathcal{F}(a_4) \cap \mathcal{T}(a_3) = \varnothing$$

Example run

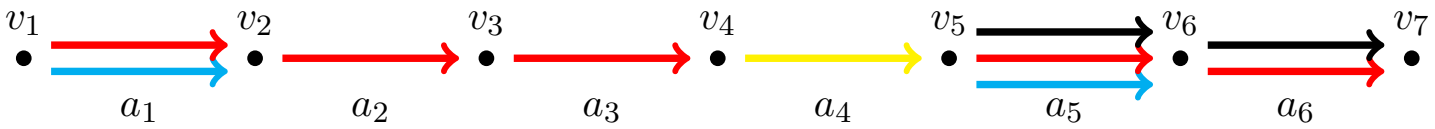


Figure: Computing $\mathcal{T}(\mathcal{P})$

$$\mathcal{T}(a_5) = \mathcal{F}(a_5) \texttt{ since } \mathcal{F}(a_5) \cap \mathcal{T}(a_4) = \varnothing$$

Example run

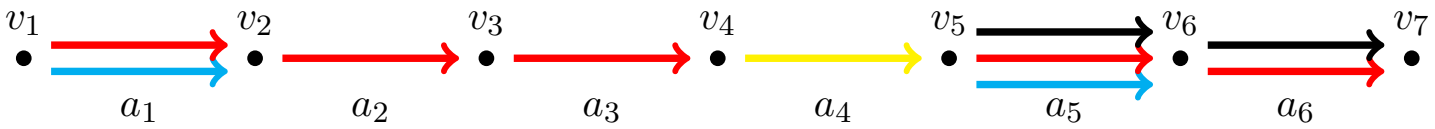


Figure: Computing $\mathcal{T}(\mathcal{P})$

$$\mathcal{T}(a_6) = \mathcal{F}(a_6) \cap \mathcal{T}(a_5) \texttt{ since not empty}$$

Example run

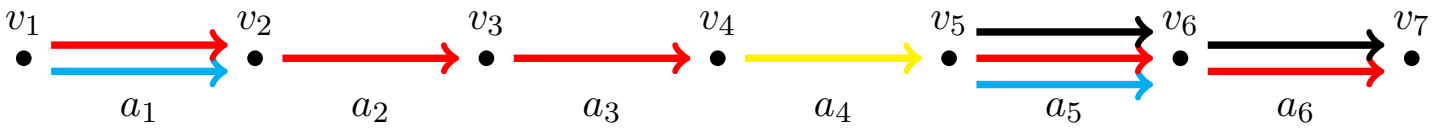


Figure: Computing $\mathcal{T}(\mathcal{P})$

Start to compute $\mathcal{H}(\mathcal{P})$

Example run

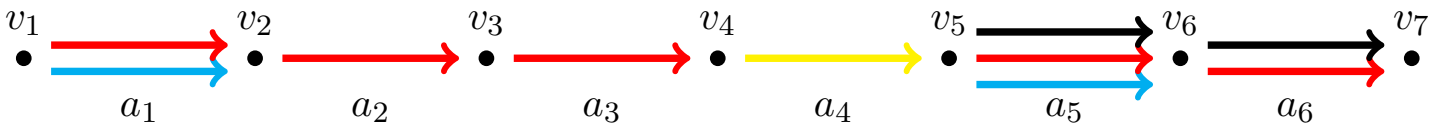


Figure: Computing $\mathcal{H}(\mathcal{P})$

$$\mathcal{H}(a_6) = black$$

Example run

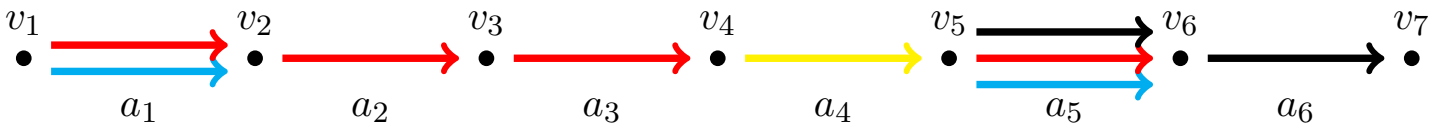


Figure: Computing $\mathcal{H}(\mathcal{P})$

$$\mathcal{H}(a_6) = black$$

Example run

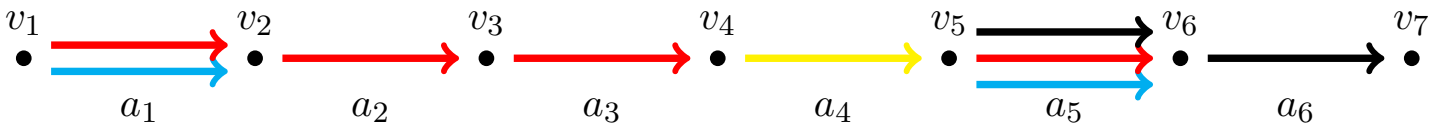


Figure: Computing $\mathcal{H}(\mathcal{P})$

$$\mathcal{H}(a_5) = black \texttt{ since } black \in \mathcal{T}(a_5)$$

Example run

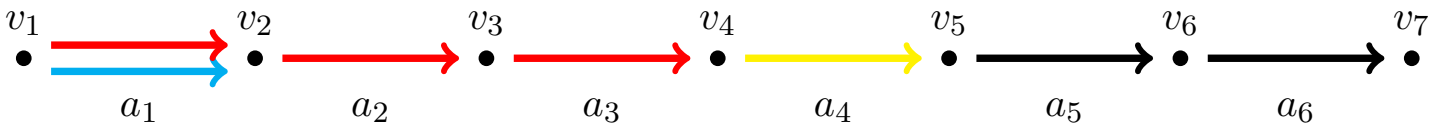


Figure: Computing $\mathcal{H}(\mathcal{P})$

$$\mathcal{H}(a_5) = black \ \texttt{since} \ black \in \mathcal{T}(a_5)$$

Example run

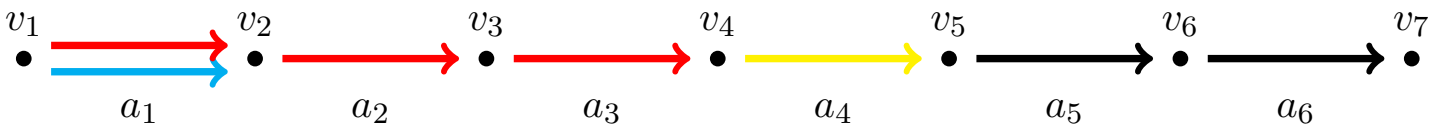


Figure: Computing $\mathcal{H}(\mathcal{P})$

Nothing to do for a_4, a_3 and a_2 since they only have 1 color

Example run

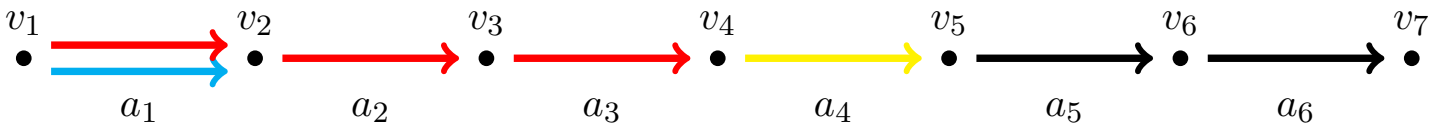


Figure: Computing $\mathcal{H}(\mathcal{P})$

$$\mathcal{H}(a_1) = \mathcal{H}(a_2) \ \texttt{since} \ red \in \mathcal{T}(a_1)$$

Example run

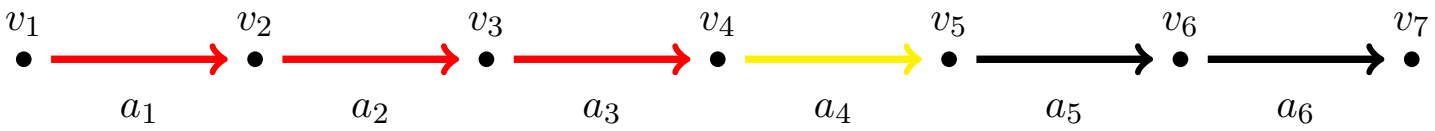


Figure: Minimum cost assignation

Example run

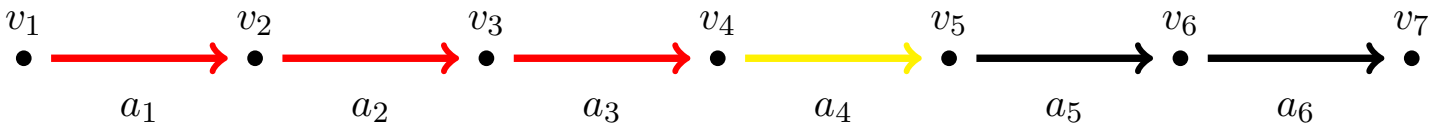


Figure: Minimum cost assignation

$$w(\mathcal{P}) = 2$$

Proof sketch

Algo Part 1.

Induction proof on the length k of $\mathcal{P}$.

□

Proof sketch

Algo Part 1.

Induction proof on the length k of $\mathcal{P}$.
If $k = 1$ then $w(\mathcal{P}) = 0$ which is optimal. □

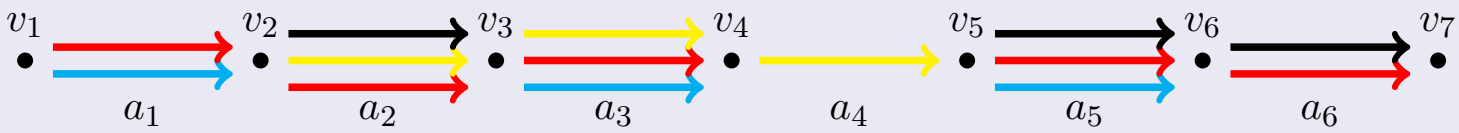
Proof sketch

Algo Part 1.

Induction proof on the length k of $\mathcal{P}$.

We suppose the algo to be true for an arbitrary length k.

If $\mathcal{F}(a_k) \cap \mathcal{F}(a_{k+1}) = \varnothing$



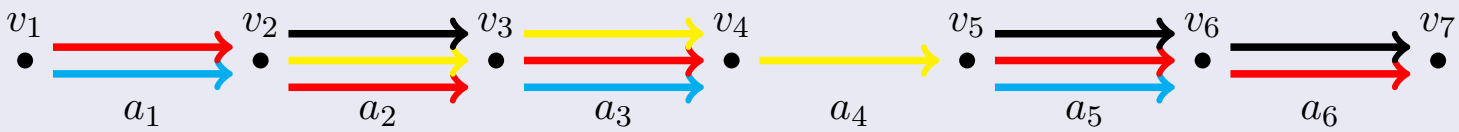
Proof sketch

Algo Part 1.

Induction proof on the length k of $\mathcal{P}$.

We suppose the algo to be true for an arbitrary length k.

If $\mathcal{T}(a_k) \cap \mathcal{F}(a_{k+1}) \neq \varnothing$



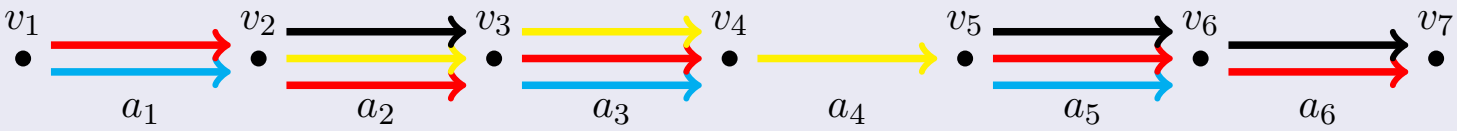
Proof sketch

Algo Part 1.

Induction proof on the length k of $\mathcal{P}$.

We suppose the algo to be true for an arbitrary length k.

If $\mathcal{T}(a_k) \cap \mathcal{F}(a_{k+1}) = \varnothing$ and $\mathcal{F}(a_k) \cap \mathcal{F}(a_{k+1}) \neq \varnothing$



Proof sketch

Algo Part 1.

Induction proof on the length k of $\mathcal{P}$.
Done □

Algo Part 2.

The number of *CS* inside $\mathcal{T}$ is the same as the number of *CS* inside $\mathcal{H}$. □

Time Complexity

The algo is made by two sub-procedures:

Recall the first part:

- $\mathcal{T}(a_1) = \mathcal{F}(a_1)$
- $\mathcal{T}(a_i) = \mathcal{F}(a_i) \cap \mathcal{T}(a_{i-1})$ `if not empty else` $\mathcal{F}(a_i)$

Complexity:

- First part : $\mathcal{O}(k * |\mathcal{C}|)$

Time Complexity

The algo is made by two sub-procedures:

Recall the second part:

- $\mathcal{H}(a_k) =$ `a rnd elt from` $\mathcal{T}(a_k)$
- $\mathcal{H}(a_i) = \mathcal{H}(a_{i+1})$ `if it is in` $\mathcal{T}(a_i)$ `else` $\mathcal{T}(a_i)$`.peek()`

Complexity:

- First part : $\mathcal{O}(k * |\mathcal{C}|)$
- Second part : $\mathcal{O}(k * \log |\mathcal{C}|)$

Time Complexity

Complexity:

- First part : $\mathcal{O}(k * |\mathcal{C}|)$
- Second part : $\mathcal{O}(k * \log |\mathcal{C}|)$

Global complexity: $\mathcal{O}(k * |\mathcal{C}|)$.
This complexity is optimal wrt the entry of the problem.

Minimize *CS* in Graph

Strategy: Use the *MDD* data structure

A state of a *MDD* is:

$$\{\text{name: String, cost: Int, colors: Set of Colors}\}$$

Algorithm

- The root $= \{\text{name: } s,\ \text{cost: } 0,\ \text{colors: } \mathcal{C}\}$

Algorithm 1: Construction of the layer $\mathcal{L}_{i+1}$

for $\forall st \in \mathcal{L}_i$ **do**
  for $\forall v \in$ `succ(st.name)` **do**
    `col` $\leftarrow \mathcal{F}($`st.name`$, v)$;
    `inter` $\leftarrow$ `col` $\cap$ `st.colors`;
    if `inter` $= \varnothing$ **then**
      $\mathcal{L}_{i+1}.add(\{\text{name: } v,\ \text{cost: } \texttt{st.cost} + 1,\ \text{colors: } \texttt{col}\})$
    else
      $\mathcal{L}_{i+1}.add(\{\text{name: } v,\ \text{cost: } \texttt{st.cost},\ \text{colors: } \texttt{inter}\})$
    end
  end
end

MDD reduction

Let s_1 and s_2 two state on the same layer $\mathcal{L}$, having same name.

- *Dominated states:* s_1 dominates s_2 if the cost of s_1 is smaller than the cost of s_2
- *s-compatible states:* s_1 and s_2 are s-compatible if they have same cost. In this case, s_1 and s_2 are removed from $\mathcal{L}$ and $s_3 = \{\text{name: } s_1.\text{name}, \text{ cost: } s_1.\text{cost}, \text{ colors: } s_1.\text{colors} \cup s_2.\text{colors}\}$ is added to $\mathcal{L}$

Complexity

- Each layer at most has $|V|$ states;
- The height of the *MDD* is k;

The overall complexity is therefore

$$\mathcal{O}(k * |V|^2 * |\mathcal{C}|)$$

An example

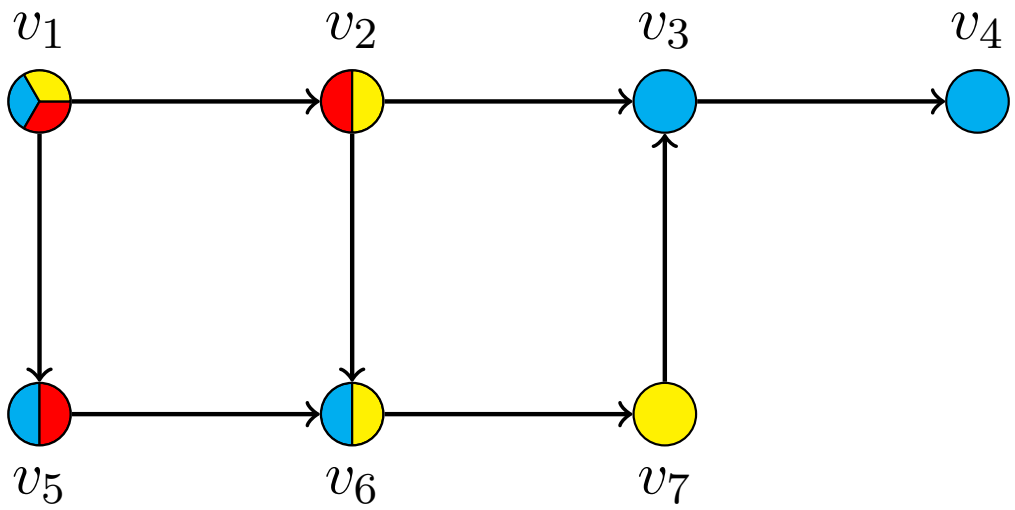


Figure: A colored graph example

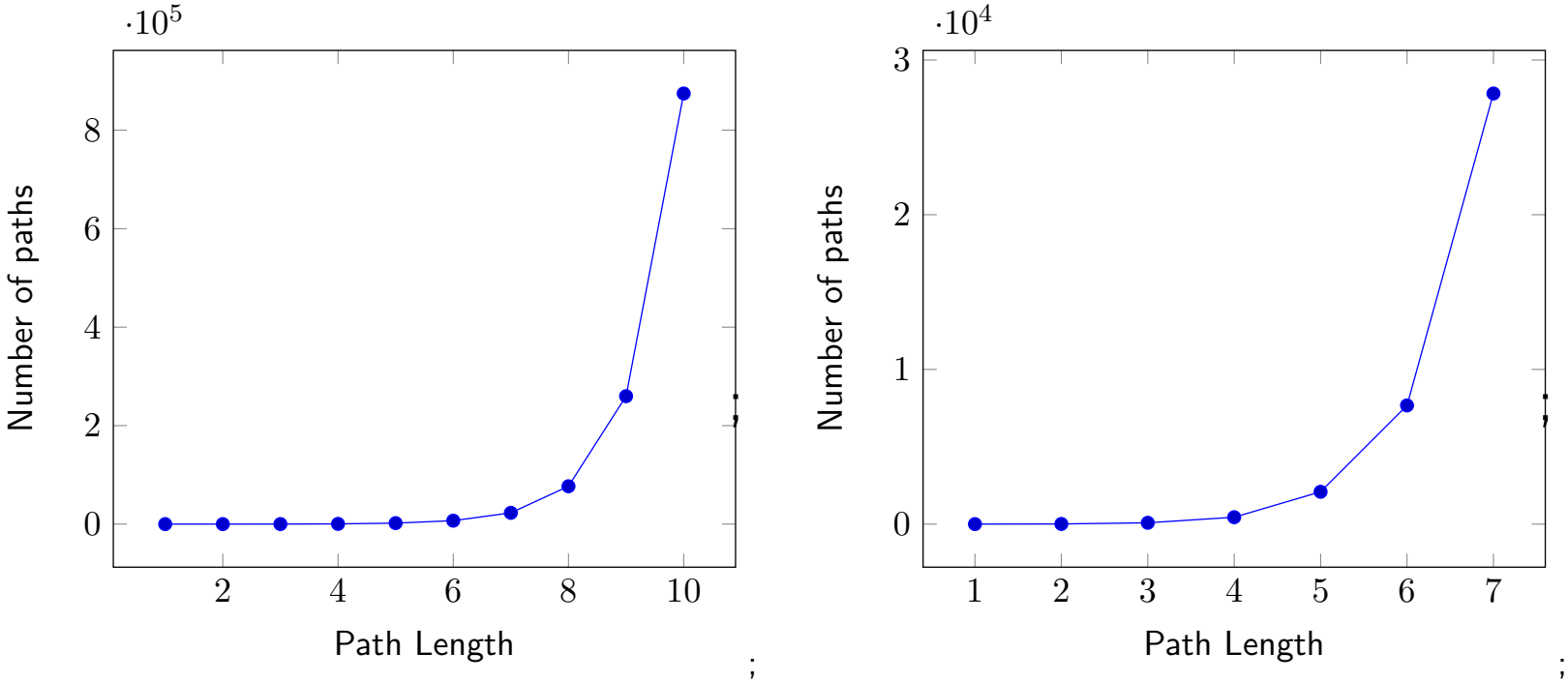
Let's take $k = 5, s = v_1$

The *allDiff* variant

A variant we can add to the problem is the introduction of the *allDiff* constraint to the nodes of the path.

- This modification entails:
  - Maintain a trace of the fathers of the current state,
  - Two states can be reduced only if they have same fathers.
- The second point causes a complexity blow up: the layer size can be $2^{|V|}$.
- The algorithm has now an exponential complexity.

Benchmark: Solutions

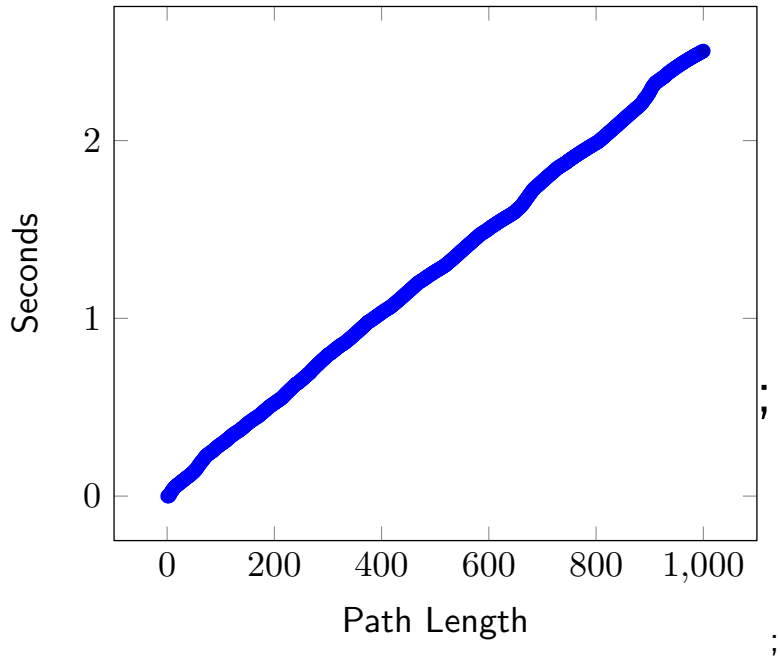


(a) Classic algorithm

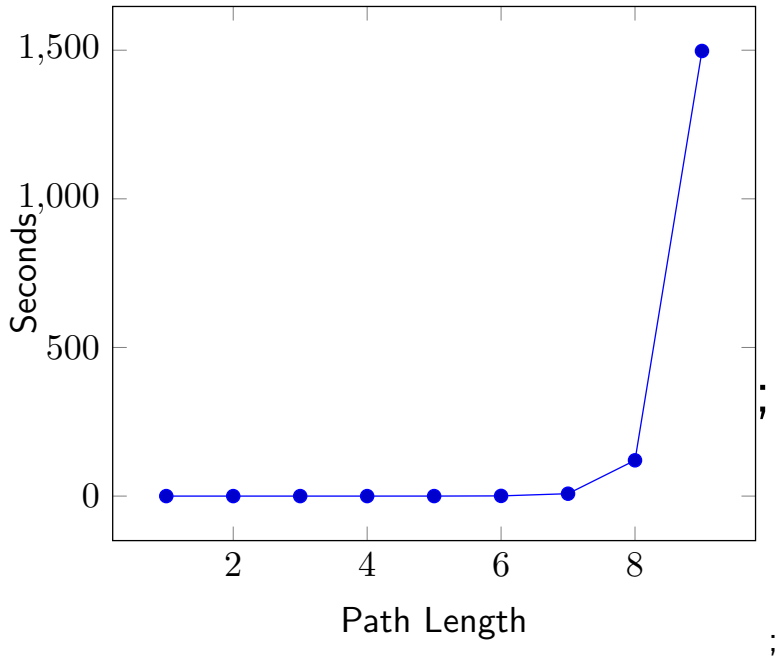
(b) *allDiff* constraint

Figure: Number of paths of a given length from the node 1

Benchmark: Time



(a) Classic algorithm



(b) *allDiff* constraint

Figure: Time taken to compute paths of a given length from the node 1

Conclusion

We have worked in order to:

- analyze of a concrete problem

Conclusion

We have worked in order to:

- analyze of a concrete problem
- give proofs for provided algorithms

Conclusion

We have worked in order to:

- analyze of a concrete problem
- give proofs for provided algorithms

Perspectives:

Conclusion

We have worked in order to:

- analyze of a concrete problem
- give proofs for provided algorithms

Perspectives:

- Add other variants such as the NValue constraint

Conclusion

We have worked in order to:

- analyze of a concrete problem
- give proofs for provided algorithms

Perspectives:

- Add other variants such as the NValue constraint
- Improve memory space saving

Conclusion

We have worked in order to:

- analyze of a concrete problem
- give proofs for provided algorithms

Perspectives:

- Add other variants such as the NValue constraint
- Improve memory space saving

Thanks!