Master in Computer Science

Course : TER

# Generation of sequences controlled by their "complexity"

*Author:*
Fissore Davide

*Supervisor:*
Jean-Charles Régin

**Abstract**

We want to generate sequences of musical "chords" (a chord is a set of notes basically) with some known constraints (allDiff, etc.) as well as control on the complexity of the sequence. This complexity in turn is defined by a dynamic programming algorithm working on the instantiated sequence, which makes the whole problem difficult.

Fall 2022

# Contents

# 1 Problem description

# 2 Definitions and notations

In the following sections $G = (V, A)$ is a directed graph where $V = (v_1, \dots, v_n)$ is the set of its vertices and $A = (a_1, \dots, a_m)$ is the set of its arcs. $n$ and $m$ represents the cardinality of rispectively $V$ and $A$. An arc $a_i \in A$ is a pair $(v_i, v_j) \in V^2$ saying that $a_i$ goes from $v_i$ to $v_j$. This arc is different from another $a_j = (v_j, v_i) \in A$.
$F$ is the coloring function taking an arc $a$ and returning the set of colors $\mathbb{C}$ associated to it. By abuse of notation we say that $F(a) = F(v_i, v_j)$ if $a = (v_i, v_j)$. $R : A \to \mathbb{N}$ is a valid affectation, that is $R(e) = c$ if and only if $c \in F(e)$. For simplicity, if $S = (a_1, \dots, a_k)$ is a list of $k$ arcs, then $F(S) = (\mathbb{C}_1, \dots, \mathbb{C}_k)$ and $R(S) = (c_1, \dots, c_k)$.
Given a path $P$ of length $k$ and its corresponding affectations $R(P)$, its weight is returned by the cost function $w(R(P))$ defined as follows:

$$w(R(P)) = \sum_{i=1}^{k-1} (c_i \neq c_{i+1})$$

$w_{OPT}(R(P))$ is the minimal weight of a path among all the possible affectation $H(P)$, this affectation is said to be optimal $R_{OPT}(P)$. Finally, we say that a shortest path from $v_i$ to $v_j$ in a graph $G$ is a path $P$ starting in $v_i$ and ending in $v_j$ whose optimal affectation $R_{OPT}$ is the minimal among all the other possible paths in $G$.

# 3 Minimize Switches in Paths

The goal of this section is to provide a greedy algorithm able to compute an optimal affectation $H$ of a given path $P$. The obtained result, will then be extended to general graphs using the **XXX matrix**.

## 3.1 Procedure

This problem can be solved through a greedy strategy: taking a path $P$ and a coloring function $F$, we must delay a color switch as much as possible. At the end we will have selected the biggest $l \in [1, k]$ such that the edges $(e_1, \dots, e_l)$ have at least one color in common. We repeat this procedure from the edge $e_{l+1}$ until reaching the end of our path. An implementation of this algorithm can be found in Algorithm 1.

## 3.2 Proof

Let $R = (c_1, \dots, c_k)$ be a solution returned by our algorithm, we can easily prove by induction on the length of the path that the solution is optimal.

For $k = 1$ we have $w(R) = 0$ by defintion of the weight function.

Let's suppose that the solution $R$ is an optimal one for every path of length at least $k$. We want to prove that the algorithm is always valid for a path of length $k + 1$, we see that:

- if $F(e_k) \cap F(e_{k+1}) = \varnothing$ then we are forced to do a color switch, for every affectation of the edge $R' = ((c_1, \dots, c_k))$. Since, by ipothesis, the affectation of the edges $w(R')$ is optimal, then it will remain optimal for any affectation of the edge $e_{k+1}$ and $w(R) = w(R') + 1$.
- if $F(e_k) \cap F(e_{k+1}) \neq \varnothing$
  - if $c_k \in F(e_{k+1})$ then the algorithm we give to $e_{k+1}$ the same color of $e_k$. This will not increase the number of color switch which will remain optimal.
  - if $c_k \in F(e_{k+1})$ then the algorithm will force a color switch even if it would have been possible to give them the same color. Despite this, if we decide to give the same colors to $e_k$ and $e_{k+1}$ then we are only anticipating a color switch, and in the end $w(R)$ will remain optimal.

### 3.3 Time Complexity

We can analyze the time complexity of this procedure from Referencesminpathalgo. We have two loops of size $k$ (the length of the path). Inside them we make intersection between sets of at most $s$ colors, then the intersection between two sets of that size will take $O(s)$. Finally, the global time complexity will be $O(2 * k * s) = O(k * s)$.

### 3.4 An example run

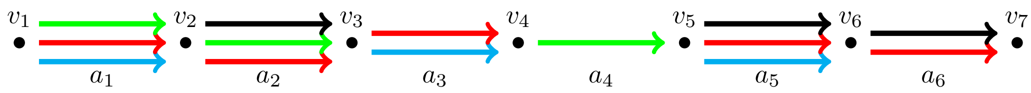


Figure 1: A path example

Let's take Figure 1, where $P = (a_1, \dots, a_6)$ and $F$ such that

$$\begin{aligned} F(P) = (&\{cyan, red, green\},\\ &\{red, green, black\},\\ &\{cyan, red\},\\ &\{green\},\\ &\{cyan, red, black\},\\ &\{red, black\}) \end{aligned}$$

The longest subpath of same color, starting from the vertex $v_1$, is $P_1 = (a_1, a_2, a_3)$ such that $R(a) = red$ for all $a \in P_1$. Then $R(e_4) = green$ and $R(a_5) = R(a_6) = black$. This affectation $H = (red, red, red, green, black, black)$ has $w(R) = 2$ and is optimal.

### 3.5 Exstention on cycles

A cycle in a path whose starting node coincide with its last one. We see that the previous algorithm is no more effective, since we have to keep into accout the potential color switch between the first and the last edge of it. Despite this, the procedure proposed in Section 3.1, can be easily modified to provide an optimal affectation on cycles. Let's take the path of Figure 1 and imagine that nodes $n_1$ and $n_7$ coincide. We now see that the affectation $H$ of Section 3.4 is no more optimal: $w(R) = 3$, while the affectation $H' = (red, red, red, green, red, red)$ as a cost of 2. In order to take into account this situation, we assign to the first $P_1$ and the last $P_l$ sub-path of edges with same colors a set of common colors. Finally if the intersection of $P_1$ and $P_l$ is not empty, we will affect them to a color they share, otherwise, whatever choice of color for $P_1$ and $P_l$ will not influence the final cost of the chosen affectation.

Concretely, take the example in Figure 1, then $P_1 = (a_1, a_2, a_3)$ and $P_l = (a_5, a_6)$. Let $C_1 = \bigcap_{a \in P_1} R(a)$ and $C_2 = \bigcap_{a \in P_2} R(a)$. We know that both $C_1$ and $C_2$ are non-empty. Then since $C_1 \cap C_2 = \{red\}$ then we can set $red$ to all arcs in $P_1$ and $P_2$ reducing therefore the overall switch number.

## 4 Minimize color switches on Directed Graphs

The previous section provides a strategy to compute the smallest cost of a given path. The key idea is to delay color switches and in this section we try to rework this algorithm in order to apply it on general directed graph. The goal is to find paths made of a fixed number of edges between two vertices in order to minimize the number of color switches.

### 4.1 Preliminaries

Floyd [1] and Warshall [2], in respectively 1959 and 1962, gave an implementation [4] of an algorithm able to compute the shortest path of a directed weighted graph.

Let $G$ be a directed graph and $c$, such that for all couple of vertices $i, j$ of $V$, if there exsits no arc going from $i$ to $j$ in $A$ then $c(i, j) = \infty$. Let $M$ be the $n \times n$ adjency matrix of $G$ such that each cell $c_{ij}$ equals $c(i, j)$. Note that for each $v \in V$, $c(v, v) = 0$.

The goal of the algorithm is to update the weight of each cell for every iteration. In particular at time 0 we know the distance from every vertex to each of its successors and to improve the informations about the global shortest path, we look for path from every couple $i, j \in V^2$ passing trhough a third vertex $k$ and take the minimum distance. We have to repeat this procedure $n$ times in order to look for paths of length at most $n$.

$$c_{ij} = \min_{k \in V}(c(i,k) + c(k,j))$$

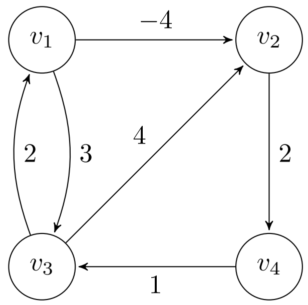


Figure 2: A directed weighted graph example

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $v_1$ | 0 | | | |
| $v_2$ | | | | |
| $v_3$ | | | | |
| $v_4$ | | | | |

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| $v_1$ | 0 | | | |
| $v_2$ | | | | |
| $v_3$ | | | | |
| $v_4$ | | | | |

Column Nuumber 1 Column Number 2 Column Number 2

Let's take for example the graph of Figure 2. The corresponding matrix

### 4.2 Matrix Method

### 4.3 MDD strategy

## 5 Simple Paths

### 5.1 Preliminaries

## 6 NValue Constraint

## 7 Conclusion

## 8 References

[1] Robert W. Floyd. *Algorithm 97*. 1959.

[2] Stephen Warshall. *A Theorem on Boolean Matrices*. 1962.

[3] Mostafa Haghir Chehreghani. *Effectively Counting s-t Simple Paths in Directed Graphs*. Report. Teheran Polytechnic, 2022.

[4] *Floyd–Warshall algorithm*. 2022. URL: https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm.

[5] *Generalized Floyd–Warshall algorithm*. 2022. URL: https://fr.wikipedia.org/wiki/Probl%C3%A8me_de_plus_court_chemin#Algorithme_de_Floyd-Warshall_g%C3%A9n%C3%A9ralis%C3%A9.

[6] *Number of paths of fixed length / Shortest paths of fixed length*. June 8, 2022. URL: https://cp-algorithms.com/graph/fixed_length_paths.html.

# A Algorithms

## A.1 Minimize color switches in a path

**Algorithm 1:** Minimize color switches in a path algorithm

```
Input: P = (a_1, ..., a_k), F := a path and the color function
Output: H := a path affectation minimizing the color switches
colSet ← [F(a_i) for i ∈ [1..k]];
for i ← 2 to k do
    inter ← colSet[i − 1] ∩ colSet[i] ;            // Delay a color switch
    if inter ≠ ∅ then
        colSet[i] ← inter;
    end
end
H ← [colSet[i].choose() for i ∈ [1..k]];
for i = k − 2 downto 1 do
    if H[i + 1] ∈ colSet[i] ∧ H[i] ≠ H[i + 1] then
        H[i] ← H[i + 1] ;            // If possible the R(e_i) equals R(e_{i+1})
    end
end
return H;
```