MASTER INFORMATIQUE

UNIVERSITÉ CÔTE D'AZUR

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 **1**

2 Definitions and notations **1**

3 Minimize Switches in Paths **1**
3.1 Procedure . . . 1
3.2 Time Complexity . . . 2
3.3 An example run . . . 2
3.4 Extension on cycles . . . 3

4 Minimize color switches with matrices **3**
4.1 Floyd-Warshall algorithm . . . 3
4.2 Paths of fixed length with minimum cost . . . 4
4.3 Minimize color switches with matrices . . . 4

5 Minimize colors switches with MDD **5**
5.1 Multi-Valued Decision Diagram . . . 5
5.2 MDD strategy . . . 6
5.3 The all different constraint . . . 6
5.4 Find simple paths . . . 6
5.5 The NValue constraint . . . 6

6 My implementation **6**
6.1 Algorithm comparison . . . 6

7 Conclusion **6**

8 References **6**

A Algorithms **7**
A.1 Minimize color switches in a path . . . 7

1 Problem description

2 Definitions and notations

In this section we fix some notations that will be reused next.
$G = (V, A)$ is a directed graph where $V = (v_1, \ldots, v_n)$ is the set of its vertices and $A = (a_1, \ldots, a_m)$ is the set of its arcs. n and m represents the cardinality of respectively 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. The arc (v_i, v_j) is different from (v_j, v_i).
colF is the coloring function and takes an arc a as parameters. It returns the set of colors $\mathbb{C}$ associated to a. By abuse of notation we say that $\text{colF}(a) = \text{colF}(v_i, v_j)$ if $a = (v_i, v_j)$.
$R : A \to \mathbb{N}$ is the affectation function, and $R(e) = c$ if $c \in \text{colF}(e)$. For simplicity, if $P = (a_1, \ldots, a_k)$ is a list of consecutive arcs whose length is k, then $\text{colF}(P) = (\text{colF}(a_1), \ldots, \text{colF}(a_k)) = (\mathbb{C}_1, \ldots, \mathbb{C}_k)$ and $R(P) = (R(a_1), \ldots, R(a_k)) = (c_1, \ldots, c_k)$.
Given a path P of length k and its corresponding affectations $H = R(P)$, the weight of H is the number of its color switches. This cost is computed by $w(R(P))$.

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

$w_{OPT}(H)$ is the minimal weight of a path among all the possible affectation H of P, R_{OPT} is an optimal affectation.
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 matrix technique proposed in Section 4.3 or the MDD strategy of Section 5.2.

3.1 Procedure

This problem can be solved through a greedy strategy: taking a path P and a coloring function colF, we delay a color switch as much as possible. Tha algorithm is decomposed in two main parts where the first one affect each arc a_i to a subset of colors of $\text{colF}(a_i)$ and a second one which make a unique affectation for each arc.

Procedure part 1. We start from the colors associated to the first arc and we take the colors of the second arc in order to make the intersection of them. We repeat this operation for all the following arcs until the intersection returns a non-empty set or we do not reach the end of the path. At the i^{th} arc, if the intersection set is empty, we make a color switch. Then we repeat the same procedure from this arc starting with a set of colors equal to $\text{colF}(a_i)$. This procedure returns for each arc a subset of its associated colors.

Procedure part 2. We have to make a unique affectation, this is possible by reading the list of set returned by the previous part of the algorithm. For the last arc we take one among all the possible colors. For the next subset of colors related to the before-last arc, we take the same color of the last arc if present, otherwise a random color from its set. We repeat this operation until reaching the first arc of the path.

An implementation of this procedure can be found in Algorithm 2.

Proof (First part of the procedure). Let $H_{\mathbb{C}} = (\mathbb{C}_1, \ldots, \mathbb{C}_k)$ be a solution returned by the first part of our algorithm, we prove by induction on the length of the path that $H_{\mathbb{C}}$ minimizes the number of color switches. After this first proof, we will show that the number of color switches returned by the second part of the algorithm is the same as the one returned by the first part.
By definition of the weight function, if $k = 1$ we have $w(H_{\mathbb{C}}) = 0$ which is the optimal cost.
Let's suppose that $H_{\mathbb{C}}$ is an optimal solution for every path of length at least k. We want to prove that the the new affectation $H'_{\mathbb{C}}$ returned by the algorithm for a path of length $k+1$ is still optimal. We have to analyze two main situations:

- if $\text{colF}(a_k) \cap \text{colF}(a_{k+1}) = \varnothing$ then, for any color c_i chosen from $\text{colF}(a_{k+1})$ $w(H'_{\mathbb{C}}) = w(H_{\mathbb{C}}) + 1$, *i.e.* a color switch is forced.
- otherwise, if $\text{colF}(a_k) \cap \text{colF}(a_{k+1}) \neq \varnothing$ we have two sub-cases to treat:
  - if it exist a subset of colors $\mathbb{C}_{k+1} \subseteq \text{colF}(a_{k+1})$ which is included in $\mathbb{C}_k$, *i.e.* there exists at least a color in $\text{colF}(a_{k+1})$, allowing us to have $w(H'_{\mathbb{C}}) = w(H_{\mathbb{C}})$.
  - this final case is the most interesting to treat because we have $\mathbb{C}_k \cap \text{colF}(a_{k+1}) = \varnothing$ but of the other hand we have $\text{colF}(a_k) \cap \text{colF}(a_{k+1}) \neq \varnothing$. It means that we have done a particular choice of colors associated to the arc a_k causing a color switch ($w(H'_{\mathbb{C}}) = w(H_{\mathbb{C}}) + 1$) even if it would have been possible to affect a_k to the same color of its following arc.
    Let's suppose, by means of contradiction, that it exists a better affectation of subsets for the subpath $(a_1, \ldots, a_{k+1})$. Without loss of generality, let's suppose that the intersection of the first $\text{colF}(a_i), 0 \leqslant i \leqslant k$ is not empty, *i.e.* there exists a color which is shared by all the a_i first arcs. The cost of the this subpath is 0 since all edges can have the same color. If we want to add the new edge $ak + 1$ to the path without increasing the number of color switch we need that there exists at least a colors in $\text{colF}(ak + 1)$ which is common to all the $\text{colF}(a_i), 0 \leqslant i \leqslant k$ first edges. However, this is not possible, otherwise the algorithm would have kept this color as a valid option for all the path. A contradiction.

We can conclude that the number of color switches returned by the first part of the procedure is minimal. □

Proof (Second part of the procedure). In the previous proof, we have shown that the number of color switches return by the first part of the algorithm is minimal. We only have to prove that its second part return exactly the an affectation with the sae number of colors switches to conclude that our algorithm is optimal.
Let $(\mathbb{C}_1, \ldots, \mathbb{C}_k)$ be the subset affectation returned by the previous part of the algorithm. Note that, by construction of the first algorithm, for each set $\mathbb{C}_i$, its successor $\mathbb{C}_{i+1}$ is either a subset of $\mathbb{C}_i$ or $\mathbb{C}_i \cap \mathbb{C}_{i+1} = \varnothing$. Starting from the last arc of the path, we can choose an arbitrary color $c_k \in \mathbb{C}_k$ for a_k. Then for the edge a_{k-1}, we choose the same color of a_len if possible and repeat the same procedure until reaching the first edge of the path.
By definition, the number of color switches of the generated path will be qual to the number of two by two different $i, j \in [0..k]$ such that $\mathbb{C}_i \cap \mathbb{C}_{i+1} = \varnothing$. □

3.2 Time Complexity

We can analyze the time complexity of this procedure from the implementation proposed in Algorithm 2. 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 $\mathcal{O}(s)$. Finally, the global time complexity will be $\mathcal{O}(2 * k * s) = \mathcal{O}(k * s)$.

3.3 An example run

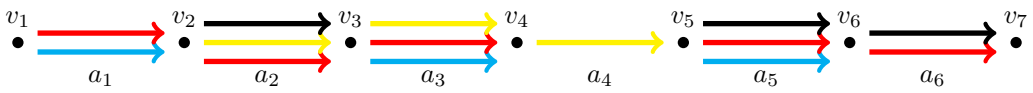

Figure 1: A path example

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

$$\begin{aligned}\text{colF}(P) = (&\{cyan, red\}, \{red, yellow, black\},\\ &\{cyan, red, yellow\}, \{yellow\},\\ &\{cyan, red, black\}, \{red, black\})\end{aligned}$$

Here we give a solution of how the procedure proposed in Section 3.1 would solve it. The first procedure will return a list of subsets equal to

$$\begin{aligned}H' = (&\{cyan, red\}, \{red\},\\ &\{red\}, \{yellow\},\\ &\{cyan, red, black\}, \{red, black\})\end{aligned}$$

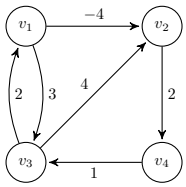

Figure 2: A directed weighted graph example

Then the second part of the algorithm would return an optimal solution which is, in this case, $H = (red, red, red, yellow, black, black)$, with $w(H) = 2$.

One can note that there can exist other optimal affectations, from Figure 1 we can choose $H_2 = (cyan, yellow, yellow, yellow, red, red)$, but in any case, any other affectations will not be less than $w(H)$.

3.4 Extension on cycles

A cycle in a path whose starting node coincide with its last one. In this situation, the previous algorithm is no more effective, since we need to keep into account the potential color switch between the first and the last arcs. However, 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. The affectation H of Section 3.3 is no more optimal since $w(H) = 3$, whereas the cost of the affectation $H' = (red, red, red, yellow, red, red)$ is 2.

In order to consider this situation, it is important to look at the intersection between the first and the last set of colors returned by the first part of the procedure. In particular, the subset associated to the extrema arcs of the path, will be modified into the intersection of their corresponding sets if non-empty. Finally we can apply second part of the algorithm.

Concretely, take the example in Figure 1, we intersect $R'(a_1)$ with $R'(a_7)$. Since this intersection is non-empty, then $R'(a_1) = R'(a_7) = \{red\}$. And the resulting affectation will be exactly H'.

4 Minimize color switches with matrices

The previous section provides a strategy to compute the smallest cost of a given path. It has been shown that an optimal strategy is to delay color switches as mush as possible. In this section we reuse this concept in order to find paths with a *fixed* number of edges between two vertices minimizing the number of color switches.

4.1 Floyd-Warshall algorithm

Floyd [1] and Warshall [2], in respectively 1959 and 1962, gave an implementation [6] of an algorithm able to compute the shortest path between any pair of vertices of a directed weighted graph. The solution is found in polynomial time over the number of vertices of the graph.

In particular, let G be a directed graph and a cost function w, such that for all pair of vertices i, j of V, if there exists no arc going from i to j in A then $w(i, j) = \infty$ and for each $v \in V$, $w(v, v) = 0$. Let M be the $n \times n$ adjacency matrix of G such that each cell M_{ij} equals $w(i, j)$.

The goal of the algorithm is to build a new matrix N whose cells contains the weight of the shortest path for every pair of vertices. This matrix is updated iteratively: at time 0, we have $N^0 = M$ representing all the shortest path of length *at most* 1 between two vertices. This information should however be improved to find out paths of smaller length made of more than one arc. Therefore, for every pair $i, j \in V^2$, we seek if a shortest path from i to j passing through a third vertex k exists.

$$N_{ij} = \min_{k \in V}(N_{ik} + N_{kj}) \tag{2}$$

The second iteration, N^2, computes all the shortest paths of length *at most* 2 for every pair of vertices. Globally, the matrix should be updated n times since, except for negative cycles, every shortest path between two vertices will pass through every vertex *at most* one time.

The overall time complexity of the Floyd-Warshall algorithm is $\mathcal{O}(n^3)$, we need to loop n times through a $n \times n$ matrix.

	v_1	v_2	v_3	v_4
v_1	0	-4	3	∞
v_2	∞	0	∞	2
v_3	2	4	0	∞
v_4	∞	∞	1	0

(a) Iteration 1

	v_1	v_2	v_3	v_4
v_1	0	-4	3	-2
v_2	∞	0	∞	2
v_3	2	-2	0	0
v_4	∞	∞	1	0

(b) Iteration 2

	v_1	v_2	v_3	v_4
v_1	0	-4	3	-2
v_2	∞	0	∞	2
v_3	2	-2	0	0
v_4	3	-1	1	0

(c) Iteration 3

	v_1	v_2	v_3	v_4
v_1	0	-4	-1	-2
v_2	5	0	3	2
v_3	2	-2	0	0
v_4	3	-1	1	0

(d) Iteration 4

Table 1: Floyd-Warshall algorithm execution of Figure 2

Floyd-Warshall algorithm run Let's take the directed graph represented in Figure 2. The corresponding matrix M is indicated in Table 1a. At the iteration 4, the distance from the vertex v_1 to vertex v_3 is updated to -1 since there is a shorter path going from n_1 to v_4 and then from v_4 to v_3. Its overall cost is given by $c_{1,4} + c_{4,3} = -2 + 1 = -1$ which is less than the direct path v_1 to v_3.

4.2 Paths of fixed length with minimum cost

As explained in [7] and [8], the Floyd-Warshall algorithm can be generalized in order to compute shortest paths on directed weighted graphs having a *fixed* number of edges. This approach is based on the theory of semirings [3].

Semiring A *semiring*[9] is a algebraic structure composed by a set of elements R and two binary operators $\oplus$ and $\otimes$. $(R, \oplus)$ forms a commutative monoid with an identity element z. $(R, \otimes)$ forms a monoid with an identity element called e, it is right distributive over $\oplus$ and z is absorbing over $\otimes$. A semiring differs from a ring because it is not required to have an inverse for the $\oplus$ operation.

Floyd-Warshall generalized algortithm Let M be the adjacency matrix of a graph whose cells on the diagonal have infinity weights if there is no self-loop on the considered vertex. We say that N^k is the matrix where each cell N_{ij} contains the cost of the shortest path from i to j with *exactly* k edges[1]. The update function of this generalized approach differs from Equation (2) since the cost of the cell c_{ij} at time k will depend of the cost at the previous iteration and the adjacency matrix.

$$N_{ij}^k = \min_{v \in V}(N_{iv}^{k-1} + M_{vj}) \tag{3}$$

The time complexity of this computation is $\mathcal{O}(n^3 k)$, since to pass from N^i to N^{i+1} we must read n time the $n \times n$ matrix and globally the matrix is updated k times.

Link with semirings It is possible to rewrite this equation in a more concise way using the definition of semiring. In fact, if, from Equation (3), the min operator is the $\oplus$ and the $+$ operator is the $\otimes$. We have that $N^k = \oplus(N^{k-1} \otimes M)$. We can further simplify the notation and say that $N^k = N^{k-1} \odot M = M^{\odot k}$. Finally, since min and $+$ are associative, we can improve the previous complexity using the binary exponentiation [4] and get $\mathcal{O}(n^3 \log k)$.

4.3 Minimize color switches with matrices

In this section we propose an adaptation of the generalized Floyd-Warshall algorithm in order to compute shortest paths of fixed length minimizing the number of color switches in oriented graphs. This adaptation wants to merge this procedure with the idea of delaying color switches proposed in Section 3.1.

The adjacency matrix M is defined differently, since we do not have exact costs associated to arcs: the cost depends on the color of two adjacent color affectations. In our implementation, M is represented by the coloring function where $\text{colF}(ij) = \varnothing$ if there is no edge between the two vertices i and j.

The cells of the N^0 matrix is a pair *(w, cols)* where: w is the cost of the path and *cols* is the set of colors minimizing the number of color switches for the path going for each vertex v_i to v_j.

Similarly to the matrix computation illustrated in the previous section, M^k depends on the matrix at time $k-1$ and the coloring function (*i.e.* the modified adjacency matrix). For all $i, j \in V$, $N_{ij}^0 = \{w \leftarrow 0 \text{ if } \text{colF}(ij) \neq \varnothing;\ cols \leftarrow \text{colF}(ij)\}$. The N^{k+1} is computed by Algorithm 1.

[1] Note that in Section 4.1 we spoke about path of *at most* k edges.

Algorithm 1: Compute N^{k+1}

Input: N^k, colF, respectively, the matrix at time k and the coloring function
Output: N^{k+1} the matrix at time $k+1$

$n \leftarrow$ the number of vertices of the graph;
// Matrix initialization
$N^{k+1} \leftarrow$ new $n \times n$ matrix ;
$\forall i, j \in [0..n]^2 : N_{ij}^{k+1} \leftarrow \{w \leftarrow \infty;\ cols \leftarrow \varnothing\}$;
// Procedure start
for $i = 1$ **to** n **do**
  for $j = 1$ **to** n **do**
    for $p = 1$ **to** n **do**
      $\mathcal{I} \leftarrow N_{ip}^k.cols \cap \text{colF}(pj)$;
      $cost \leftarrow N_{ip}^k.w + (\text{if } \mathcal{I} = \varnothing \text{ then 1 else 0})$;
      $\mathcal{S} \leftarrow (\text{if } \mathcal{I} = \varnothing \text{ then colF}(pj) \text{ else } \mathcal{I})$;
      if $cost < N_{ij}^{k+1}.w$ **then**
        $N_{ij}^{k+1} \leftarrow \{w \leftarrow cost;\ cols \leftarrow \mathcal{S}\}$;
      else if $cost = N_{ij}^{k+1}.w$ **then**
        $N_{ij}^{k+1}.cols \leftarrow \mathcal{S} \cup N_{ij}^{k+1}.cols$;
      end
    end
  end
end
return N^{k+1};

Analyze of Algorithm 1 The first step of the algorithm is to initiate the resulting matrix N^{k+1} such that each cell as an empty set of colors and an infinity cost. After this initialization, we loop over each pair of vertices ij and, as for the generalized version of the Floyd-Warshall algorithm, we look for minimal paths passing through each vertex $p \in V$. This distance is obtained following the w function: if the intersection $\mathcal{I}$ of colF(pj) and the color set of cell ip of N^k is not empty, we are able to avoid a color switch and therefore, the cost of the path ip, pj is the same as the cost of the path ip. On the other hand, if the intersection is empty, the cost of the path will be 1 more than the cost of the path ip. $\mathcal{S}$ is the set of colors that can be associated to the arc pj. It is equal to $\mathcal{I}$ if $\mathcal{I}$ is non-empty (*i.e.* we are delaying the color switch), otherwise, it will be affected to colF(pj), since any color in colF(pj) will not avoid a color switch.

Let $0 \leq p' \leq p \leq n$. While looping over all the intermediate vertex v_p, we can have three possible scenarios:

- there exists a path passing through $v_{p'}$ which is less than the path passing through v_p, the path through n_p can be ignored;
- the computed cost is strictly less than all the the previous path passing through $v_{p'}$, in this case the shortest path between ij will take this new cost and its set of colors will $\mathcal{S}$;
- finally, the cost passing through v_i equals a previous minimal one. In this case, the cost is not updated, but the colors we can give to the edge ij, to reduce the possibility of a switch, will be the union of the colors of the previous best affectations and $mathcalS$.

Add binary exponentiation

5 Minimize colors switches with MDD

5.1 Multi-Valued Decision Diagram

A *Multi-Valued Decision Diagram* is

5.2 MDD strategy

5.3 The all different constraint

5.4 Find simple paths

5.5 The NValue constraint

6 My implementation

6.1 Algorithm comparison

7 Conclusion

8 References

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

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

[3] John E. Hopcroft et Jeffrey D. Ullman Alfred V. Aho. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Jan. 11, 1974, pp. 195–206. ISBN: 978-0-201-00029-0.

[4] *Binary Exponantiation*. June 8, 2022. URL: https://cp-algorithms.com/algebra/binary-exp.html.

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

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

[7] *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.

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

[9] *Semiring*. Oct. 6, 2022. URL: https://en.wikipedia.org/wiki/Semiring.

A Algorithms

A.1 Minimize color switches in a path

Algorithm 2: Shortest path of a path

Input: $P = (a_1, \ldots, a_k)$, colF := a path and the color function
Output: H := a path affectation minimizing the color switches

```
1  colSet ← [colF(a_i) for i ∈ [1..k]];
2  for i ← 2 to k do
3  |  inter ← colSet[i − 1] ∩ colSet[i] ;                    // Delay a color switch
4  |  if inter ≠ ∅ then
5  |  |  colSet[i] ← inter;
6  |  end
7  end
8  H ← [colSet[i].choose() for i ∈ [1..k]];
9  for i = k − 2 downto 1 do
10 |  if H[i + 1] ∈ colSet[i] ∧ H[i] ≠ H[i + 1] then
11 |  |  H[i] ← H[i + 1] ;                  // If possible the R(e_i) equals R(e_{i+1})
12 |  end
13 end
14 return H;
```