Algorithmique appliquée
Projet UNO

Paul Dorbec, Cyril Gavoille

The aim of this project is to encode a program as efficient as possible to find the best sequence of cards that can be played in a hand of UNO. We first recall quickly the rules (simplified) of UNO. Cards bear a number and are of some color. A first card is chosen (here by the player), and then every card played must be of the same color or bear the same number as the previous card.

It is proven that this problem is NP-hard, with a reduction from the problem of finding a longest path in a cubic graph (see that paper).

In this project, we try to find an algorithm a efficient as possible that would solve the problem.

1 Brute force : backtracking

We first consider backtracking algorithms (see Wikipedia).

The idea is to propose a recursive algorithm. Suppose that you have already chosen a (possibly empty) beginning of a sequence. For each card that can be possibly chosen to cover the current last card, select it, look recursively for the best sequence using that card. Select the best sequence among all this sequences and return it.

This is a simple algorithm, that has a very high complexity in the worst case. Indeed, if all cards are of the same color, after picking each cards, you may select all the other cards. So possibly, you try all the permutations of the list of cards, that is $n! = \Theta(n^n\sqrt{n})$.

2 First algorithm based on dynamic programming

The idea of dynamic programming is to compute some partial optimal solution and to extend it step by step to the solution of the whole problem.

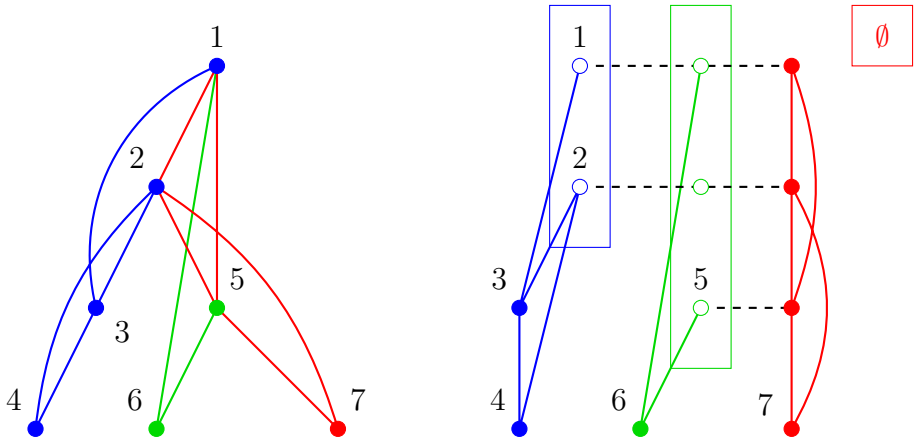

FIGURE 1 – DFS on a graph and the sets obtained, together with their intersections.

Here, we first use a separation of the deck of cards into some maximal sequences obtained within a depth first search algorithm DFS. Explore the whole card set with a DFS. All the paths from the root to a leaf are maximal paths. They form a sequence of subsets of cards. A sequence that uses two cards within different consecutive subsets must go through the intersection of these subsets to jump from one subset to the next one. For example, in Fig. 1, it is not possible to join the node with label 4 to the node ith label 6 without going through 1 or 2. We will use this to make an efficient algorithm when the intersections of the subsets are small.

Suppose we have a sequence of subsets of cards $S_1, S_2, \ldots, S_p$ (drawn with colors in Fig. 1). We consider all the possibilities of joining sequences within $\cup_{i\leq k} S_i$ with a sequence in the following. We then need to store how the sequences within $\cup S_i$ may interact with the following. What we know is that all this interactions must happen among the intersection $\cup_{i\leq k} S_i \cap \cup_{j>k} S_j$, which is the same as $S_k \cap S_{k+1}$. Thus we need to compute all the sequences within $\cup_{i\leq k} S_i$ depending on how they interact with $S_k \cap S_{k+1}$.

What does an intersection may look like ? (see Fig 2) The cards in the intersection may be connecting card, i.e. ends of subpaths. The number of subsequences with only one end in the intersection must be 0, 1 or 2 (There is no way on the rest of the card deck to join more than two half-paths into one path). The other subsequences must connect two ends in the path, so that they can be used as in a long sequence. It can also be that some cards in the intersection are visited along these sequences, thus cannot serve on the rest. So the signature of a sequence in the intersection could be :

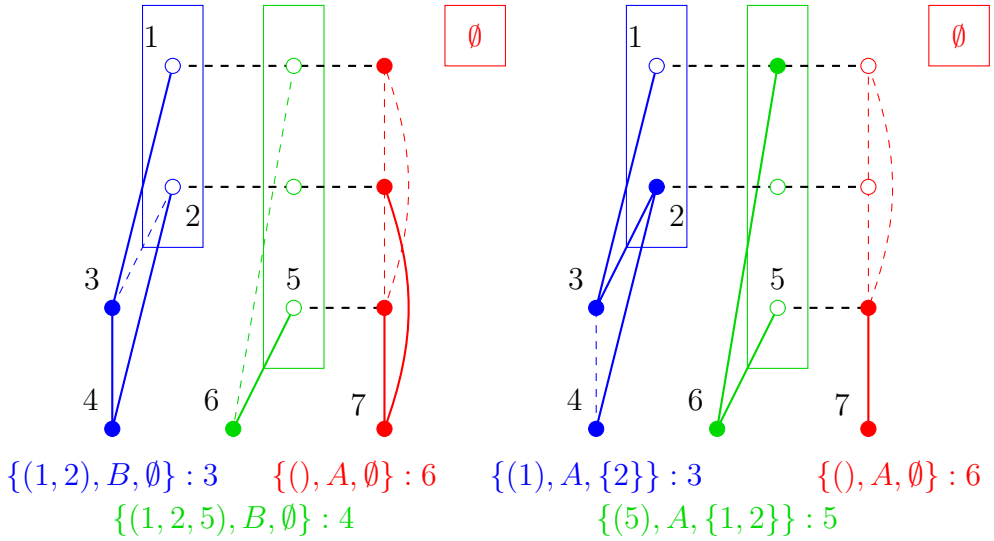

FIGURE 2 – Signatures used in the maximum path.

— an order on some cards within the intersection, i.e. a permutation of the cards in the intersection.
— whether the first card in the order is a single end in the path (type A), or whether it belongs to a pair (type B).
— for all other cards, whether they are visited or not.

For each signature, we want to store the maximal length of a sequence with that signature. Note that we don't need to store in details how this length is splitted among the different part, since if we use only part of it, we should rather consider a smaller signature. From the example of Fig. 2, the first set gives the following signatures with the length of the optimal path :

$$\{(1,2), A, \emptyset\} : 2$$
$$\{(1,2), B, \emptyset\} : 3$$
$$\{(1), A, \{2\}\} : 3$$
$$\{(1), A, \emptyset\} : 2$$
$$\{(2), A, \{1\}\} : 3$$
$$\{(2), A, \emptyset\} : 2$$
$$\{(), A, \{1,2\}\} : 3$$
$$\{(), A, \{1\}\} : 2$$
$$\{(), A, \{2\}\} : 2$$
$$\{(), A, \emptyset\} : 1$$

Now, if we have the set of maximal length for each signature within $\cup_{i \leq k} S_i$, we compute the results for all signature within $\cup_{i \leq k+1} S_i$ by a brute

force algorithm (similar to the backtrack). At the end of this algorithm, we deduce the length of the best sequence. Observe that we also need to consider the case of an empty signature if the optimal sequence does not cross the intersection.