E.A.6.10 (Cards, 2)

1.1 Modellazione

Dati i paramentri $Cards, N, M, D$ siano

- $\mathcal{I} = \{1, ..., N\}$ è l'insieme di *identificatori* per cui esiste una funzione π t.c.

$$\begin{aligned} \pi &: \mathcal{I} \to \{1, ..., D\} \\ \pi(i) &\mapsto \text{valore dell' } i\text{-esima carta} \end{aligned} \tag{1}$$

- $\mathcal{P} = \{1, ..., N\}$ è l'insieme di posizioni possibili per una carta
- $\mathrm{LP} = \left\{X_p^i \mid i \in \mathcal{I} \land p \in \mathcal{P}\right\} \cup \left\{S_p^j \mid j \in \{1,2,3\} \land p \in \mathcal{P}\right\}$ è l'insieme di lettere proposizionali t.c.
  - X_p^i è vera se la carta con id i è in posizione p
  - S_p^j è vera se il punto stazionario j è in posizione p

Il problema si può modellare con una serie di vincoli

$$\begin{aligned} \phi = {} & \phi_{\text{ALO\_pos}} \land \phi_{\text{AMO\_pos}} \land \phi_{\text{alldiff}} \land \\ & \phi_{\text{ALO\_staz}} \land \phi_{\text{AMO\_staz}} \land \phi_{\text{staz}} \land \phi_{\text{dist}} \land \\ & \phi_{\text{ord\_1}} \land \phi_{\text{ord\_2}} \land \phi_{\text{ord\_3}} \land \phi_{\text{ord\_4}} \land \neg S_1^1 \land \neg S_N^3 \end{aligned}$$

(ALO) Ogni carta ha almeno una posizione.

$$\phi_{\text{ALO\_pos}} = \bigwedge_{i \in \mathcal{I}} \bigvee_{p \in \mathcal{P}} X_p^i \tag{2}$$

(AMO) Ogni carta ha al più una posizione.

$$\phi_{\text{AMO\_pos}} = \bigwedge_{\substack{i \in \mathcal{I} \\ p_1, p_2 \in \mathcal{P} \\ p_1 < p_2}} X_{p_1}^i \to \neg X_{p_2}^i \tag{3}$$

(alldiff) Non ci sono due carte nella stessa posizione

$$\phi_{\text{alldiff}} = \bigwedge_{\substack{p \in \mathcal{P} \\ i_1, i_2 \in \mathcal{I} \\ i_1 < i_2}} X_p^{i_1} \to \neg X_p^{i_2} \tag{4}$$

(ALO) Ogni punto stazionario ha almeno una posizione.

$$\phi_{\text{ALO\_staz}} = \bigwedge_{j \in \{1,2,3\}} \bigvee_{p \in \mathcal{P}} S_p^j \tag{5}$$

(AMO) Ogni punto stazionario ha al più una posizione.

$$\phi_{\text{AMO\_staz}} = \bigwedge_{\substack{j \in \{1,2,3\} \\ p_1, p_2 \in \mathcal{P} \\ p_1 < p_2}} S_{p_1}^j \to \neg S_{p_2}^j \tag{6}$$

I punti stazionari sono posizionati in ordine. Quindi se il punto S^1 è in posizione p allora i punti successivi *non* possono essere posizionati in p o in una posizione precedente a p

$$\phi_{\text{staz}} = \bigwedge_{\substack{j_1,j_2 \in \{1,2,3\} \\ p_1,p_2 \in \mathcal{P} \\ j_1 < j_2 \\ p_1 \geq p_2}} S^{j_1}_{p_1} \rightarrow \neg S^{j_2}_{p_2} \tag{7}$$

La distanza fra i due *punti di massimo* è esattamente D

$$\phi_{\text{dist}} = \bigwedge_{\substack{p \in \mathcal{P} \\ p+D \in \mathcal{P}}} S^1_p \rightarrow S^3_{p+D} \tag{8}$$

Se il punto stazionario S^1 è in posizione p, e l'i-esima carta è in posizione q t.c. $q < p$, allora in posizione $q+1$ non ci può essere una carta di valore inferiore o uguale a $\pi(i)$

$$\phi_{\text{ord\_1}} = \bigwedge_{\substack{p,q \in \mathcal{P} \\ i,j \in \mathcal{I} \\ q < p \\ \pi(j) \leq \pi(i)}} S^1_p \wedge X^i_q \rightarrow \neg X^j_{q+1} \tag{9}$$

Simile all'Equazione 9, ma per le posizioni tra S^1 e S^2

$$\phi_{\text{ord\_2}} = \bigwedge_{\substack{p_1,p_2,q \in \mathcal{P} \\ i,j \in \mathcal{I} \\ p_1 \leq q < p_2 \\ \pi(j) \geq \pi(i)}} S^1_{p_1} \wedge S^2_{p_2} \wedge X^i_q \rightarrow \neg X^j_{q+1} \tag{10}$$

Simile all'Equazione 9, ma per le posizioni tra S^2 e S^3

$$\phi_{\text{ord\_3}} = \bigwedge_{\substack{p_1,p_2,q \in \mathcal{P} \\ i,j \in \mathcal{I} \\ p_1 \leq q < p_2 \\ \pi(j) \leq \pi(i)}} S^2_{p_1} \wedge S^3_{p_2} \wedge X^i_q \rightarrow \neg X^j_{q+1} \tag{11}$$

Simile all'Equazione 9, ma per le posizioni da S^3 in poi

$$\phi_{\text{ord\_4}} = \bigwedge_{\substack{p,q \in \mathcal{P} \\ i,j \in \mathcal{I} \\ p \leq q < N \\ \pi(j) \geq \pi(i)}} S^3_p \wedge X^i_q \rightarrow \neg X^j_{q+1} \tag{12}$$

1.2 Istanziazione

1.2.1 Parametri e variabili

$(Cards, N, M, D) = (\{1, 1, 2, 2, 3, 3, 4\}, 7, 4, 4)$

$$
\begin{aligned}
\mathrm{LP} = \{ & \\
& X_1^1, X_2^1, X_3^1, X_4^1, X_5^1, X_6^1, X_7^1, X_1^2, X_2^2, X_3^2, X_4^2, X_5^2, X_6^2, X_7^2, \\
& X_1^3, X_2^3, X_3^3, X_4^3, X_5^3, X_6^3, X_7^3, X_1^4, X_2^4, X_3^4, X_4^4, X_5^4, X_6^4, X_7^4, \\
& X_1^5, X_2^5, X_3^5, X_4^5, X_5^5, X_6^5, X_7^5, X_1^6, X_2^6, X_3^6, X_4^6, X_5^6, X_6^6, X_7^6, \\
& X_1^7, X_2^7, X_3^7, X_4^7, X_5^7, X_6^7, X_7^7, \\
& S_1^1, S_2^1, S_3^1, S_4^1, S_5^1, S_6^1, S_7^1, S_1^2, S_2^2, S_3^2, S_4^2, S_5^2, S_6^2, S_7^2, \\
& S_1^3, S_2^3, S_3^3, S_4^3, S_5^3, S_6^3, S_7^3, \\
\} &
\end{aligned}
$$

1.2.2 Vincoli

Etc… sono tanti, e si tratterebbe comunque di generarli automaticamente. Nell'encoding generato ci sono 5229 clausole…

1.3 Codifica

```rust
use crate::encoder::*;
use serde::Serialize;

#[derive(Clone, Copy, Hash, PartialEq, Eq, PartialOrd, Ord, Serialize, Debug)]
pub enum LP {
    X(usize, usize),
    S(usize, usize),
}

pub fn encode_instance(
    cards: Vec<usize>,
    cards_number: usize,
    max_value: usize,
    distance: usize,
) -> (String, Vec<LP>) {
    use Literal::Neg;

    let mut encoder = EncoderSAT::new();
    let positions = cards_number;

    // ALO_pos
    for i in 1..=cards_number {
        encoder.add((1..=positions).map(|p| LP::X(i, p).into()).collect());
    }

    // AMO_pos
    for i in 1..=cards_number {
        for p1 in 1..=positions {
            for p2 in p1 + 1..=positions {
                encoder.add(vec![Neg(LP::X(i, p1)), Neg(LP::X(i, p2))]);
            }
        }
    }

    // alldiff
    for p in 1..=positions {
        for i1 in 1..=cards_number {
            for i2 in i1 + 1..=cards_number {
                encoder.add(vec![Neg(LP::X(i1, p)), Neg(LP::X(i2, p))]);
            }
        }
    }

    // ALO_staz
    for j in 1..=3 {
        encoder.add((1..=positions).map(|p| LP::S(j, p).into()).collect());
    }

    // AMO_staz
    for j in 1..=3 {
        for p1 in 1..=positions {
```

```
            for p2 in p1 + 1..=positions {
                encoder.add(vec![Neg(LP::S(j, p1)), Neg(LP::S(j, p2))])
            }
        }
    }

    // staz
    for j1 in 1..=3 {
        for j2 in j1 + 1..=3 {
            for p1 in 1..=positions {
                for p2 in 1..=p1 {
                    encoder.add(vec![Neg(LP::S(j1, p1)), Neg(LP::S(j2, p2))])
                }
            }
        }
    }

    // dist
    for p in 1..=positions {
        if p + distance ≤ positions {
            encoder.add(vec![Neg(LP::S(1, p)), LP::S(3, p + distance).into()])
        }
    }

    // ord_1
    for p in 1..=positions {
        for q in 1..p {
            for i in 1..=cards_number {
                for j in 1..=cards_number {
                    if cards[j - 1] ≤ cards[i - 1] {
                        encoder.add(vec![
                            Neg(LP::S(1, p)),
                            Neg(LP::X(i, q)),
                            Neg(LP::X(j, q + 1)),
                        ])
                    }
                }
            }
        }
    }

    // ord_2
    for p1 in 1..=positions {
        for p2 in p1 + 1..=positions {
            for q in p1..p2 {
                for i in 1..=cards_number {
                    for j in 1..=cards_number {
                        if cards[j - 1] ≥ cards[i - 1] {
                            encoder.add(vec![
                                Neg(LP::S(1, p1)),
                                Neg(LP::S(2, p2)),
                                Neg(LP::X(i, q)),
                                Neg(LP::X(j, q + 1)),
```

```
                            ])
                        }
                    }
                }
            }
        }
    }

    // ord_3
    for p1 in 1..=positions {
        for p2 in p1 + 1..=positions {
            for q in p1..p2 {
                for i in 1..=cards_number {
                    for j in 1..=cards_number {
                        if cards[j - 1] ≤ cards[i - 1] {
                            encoder.add(vec![
                                Neg(LP::S(2, p1)),
                                Neg(LP::S(3, p2)),
                                Neg(LP::X(i, q)),
                                Neg(LP::X(j, q + 1)),
                            ])
                        }
                    }
                }
            }
        }
    }

    // ord_4
    for p in 1..=positions {
        for q in p..positions {
            for i in 1..=cards_number {
                for j in 1..=cards_number {
                    if cards[j - 1] ≥ cards[i - 1] {
                        encoder.add(vec![
                            Neg(LP::S(3, p)),
                            Neg(LP::X(i, q)),
                            Neg(LP::X(j, q + 1)),
                        ])
                    }
                }
            }
        }
    }

    encoder.add(vec![Neg(LP::S(1, 1))]);
    encoder.add(vec![Neg(LP::S(3, cards_number))]);

    encoder.end()
}
```