Dynamic Programming Algorithm

1 Overview

For simplicity, we reduce the problem of finding the best sequence of swap sequences which yields the minimum to the problem to find the minimum cost. From the dynamic programming algorithm found, it is straight-forward the solution for the former problem.

Suppose S_{ij} is the cost of satisfying the j-th dependency with the i-th permutation. Assuming it is possible (satisfying the j-th dependency with the i-th permutation), its cost will be the minimum value given by the sum of:

- the cost of solving the previous dependency $(j-1)$ with the p-th permutation, for any p;
- the cost of transforming the p-th permutation into the i-th;
- the cost of the operation that will be used (*cnot*, *r-cnot* or *l-cnot*).

Let P be the permutation set, *Swap*, *Inv* and *Lcx* be some constant and $Swaps_{pi}$ be the number of swaps in order to transform the p-th into the i-th permutation. We have that the optimal cost to solve the j-th dependency, using the i-th permutation S_{ij} is showed by Equation 1:

$$S_{ij} = \begin{cases} 0 & j = 0 \\ \min\limits_{\forall p \in P} S_{pj-1} + (Swaps_{pi} \cdot Swap) & (u,v) \in G.E \\ \min\limits_{\forall p \in P} S_{pj-1} + (Swaps_{pi} \cdot Swap) + Inv & (u,v) \in G.InvE \\ \min\limits_{\forall p \in P} S_{pj-1} + (Swaps_{pi} \cdot Swap) + Lcx & dist(u,v) = 2 \\ UNREACH & \end{cases} \quad (1)$$

2 Algorithm

Algorithm 1 shows is a pseudo-algorithm of the implementation of the description given in the previous section. Assuming that k is the number of dependencies and n is the number of vertices of the graph, follows a brief explanation of some operations:

- **Line 2:** preprocess the graph, calculating the swaps needed to transform from one permutation into another;
- **Line 3:** generates all permutations for the graph;
- **Line 4:** initializes *dyn* which is a $n! \times k$ matrix with the subproblem's solution. The first column of this matrix is initialized with 0;

- **Line 5˜16:** calculates the optimal cost from left to right, since we need the costs from the last dependency;
- **Line 12:** calculates $S_{i,tgt}$.

Algorithm 1 Exact algorithm.

Require: A directed graph $G = (V, E)$, and the program dependencies $Deps$

1: **function** SOLVE($G, Deps$)
2: $\quad SwapsMap \leftarrow constructSwapsMap(G)$
3: $\quad PermVector \leftarrow generatePermutations(G)$
4: $\quad initialize(dyn)$
5: $\quad$ **for** $i \in \{1, \dots, k\}$ **do**
6: $\qquad (a, b) \leftarrow Deps[i-1]$
7: $\qquad$ **for** $tgt \in \{0, \dots, n!\}$ **do**
8: $\qquad\quad tgtPerm \leftarrow getIthPermutation(tgt)$
9: $\qquad\quad u \leftarrow tgtPerm.get(a)$
10: $\qquad\quad v \leftarrow tgtPerm.get(b)$
11: $\qquad\quad$ **if** $(u, v) \in G.E$ **or** $dist(u, v) = 2$ **then**
12: $\qquad\qquad minVal \leftarrow CalculateMinVal(G, i, tgt, dyn)$
13: $\qquad\quad$ **end if**
14: $\qquad\quad dyn[tgt][i] \leftarrow minVal$
15: $\qquad$ **end for**
16: $\quad$ **end for**
17: $\quad$ **return** $\min_{i<n!}(dyn[i][k])$
18: **end function**

The Algorithm 2 shows the four possibilities for calculating the S_{ij} for some $i \in P$ and $j \leq k$. Lines 7˜9 get the transformation cost and lines 10˜16 add the cost of using some operation if (u, v) is not a non-reverse edge of G.

Algorithm 2 Finds the best cost to solve a dependency.

Require: A directed graph $G = (V, E)$, the dependency number i, the target permutation index tgt and the computed sub-problems dyn.

```
function CalculateMinVal(G, i, tgt, dyn)
    minVal ← {tgt, NULL, UNREACH}
    for src ∈ {0, . . . , n!} do
        srcVal ← dyn[src][i − 1]
        if srcVal.cost ≠ UNREACH then
            final ← srcVal.cost
            if tgt ≠ src then
                final ← final + (getSwapNum(src, tgt) · Swap)
            end if
            if (u, v) ∈ G.InvE then
                final ← final + Inv
            else
                if dist(u, v) = 2 then
                    final ← final + Lcx
                end if
            end if
            minVal ← min(minVal, final)
        end if
    end for
    return minVal
end function
```