NOC In Polynomial Time

Problem Statement

The problem is to find the state S^* that maximizes the objective function:

$$F(S) = \alpha(S) \cdot O(S) + \beta(S) \cdot L(S)$$

where:

- $O(S) = \sum_{\substack{i<j \\ R(i,j)=C}} (V_i I_i + V_j I_j)$, the total value from cohesive pairs,
- $L(S) = \sum_{\substack{i<j \\ R(i,j)=D}} (V_i I_i + V_j I_j)$, the total latent value from disjunctive pairs,
- $\alpha(S) = \dfrac{|\{(i,j) : R(i,j) = C \text{ or } N\}|}{M}$, and $\beta(S) = 1 - \alpha(S)$, with $M = \dfrac{n(n-1)}{2}$ being the total number of node pairs.

Key Insight

The state N (neutral) does not contribute to $O(S)$ or $L(S)$, since $\Lambda(i,j) = 0$ when $R(i,j) = N$. Assigning pairs to N dilutes the weights $\alpha(S)$ and $\beta(S)$ without adding value. Therefore, we can restrict assignments to C and D only, since any assignment including N can be improved by reassigning N-pairs to C or D.

Simplified Problem

With $E_N = \emptyset$, we have:

- $\alpha(S) = \dfrac{k}{M}$, where $k = |E_C|$ is the number of pairs assigned to C,
- $\beta(S) = \dfrac{M-k}{M}$,
- $O(S) = \sum_{(i,j)\in E_C} w_{ij}$, where $w_{ij} = V_i I_i + V_j I_j$,
- $L(S) = \sum_{(i,j)\in E_D} w_{ij}$.

Let $W = \sum_{i<j} w_{ij}$ be the total weight of all pairs. Since $E_C \cup E_D$ covers all pairs and $E_C \cap E_D = \emptyset$, we have $O(S) + L(S) = W$, so $L(S) = W - O(S)$.

Thus, the objective function becomes:

$$F(S) = \frac{k}{M}O(S) + \frac{M-k}{M}(W - O(S)) = \frac{2k-M}{M}O(S) + \frac{M-k}{M}W.$$

Optimization for Fixed k

For a fixed k, $F(S)$ depends linearly on $O(S)$:

- If $k > M/2$: $\frac{2k-M}{M} > 0$, so $F(S)$ increases with $O(S)$. To maximize $F(S)$, choose the k pairs with the largest weights.
- If $k < M/2$: $\frac{2k-M}{M} < 0$, so $F(S)$ decreases with $O(S)$. To maximize $F(S)$, choose the k pairs with the smallest weights.
- If $k = M/2$: $\frac{2k-M}{M} = 0$, so

  $$F(S) = \frac{W}{2},$$

  constant regardless of which pairs are chosen.

Polynomial-Time Algorithm

1. **Compute weights:** For each of the $M = O(n^2)$ pairs, compute $w_{ij} = V_i I_i + V_j I_j$.
2. **Sort weights:** Sort the list of M weights in $O(M \log M) = O(n^2 \log n)$ time.
3. **Precompute cumulative sums:**
    - Let $w_{(1)} \leq w_{(2)} \leq \cdots \leq w_{(M)}$ be the sorted weights.
    - Precompute $S_{\min}[k] = \sum_{i=1}^{k} w_{(i)}$ for $k = 0$ to M, with $S_{\min}[0] = 0$.
    - Compute $W = S_{\min}[M]$.
    - For any k, $S_{\max}[k] = \sum_{i=M-k+1}^{M} w_{(i)} = W - S_{\min}[M-k]$.
4. **Evaluate $F(k)$ for all k:**
    - For each k from 0 to M:
        - If $k \leq M/2$, set $T(k) = S_{\min}[k]$.
        - If $k > M/2$, set $T(k) = S_{\max}[k]$.
        - Compute
        $$F(k) = \frac{2k - M}{M} T(k) + \frac{M - k}{M} W.$$
    - This step takes $O(M) = O(n^2)$ time.
5. **Find maximum:** Identify k^* that maximizes $F(k)$, in $O(M)$ time.

The overall time complexity is dominated by sorting:

$$O(n^2 \log n),$$

which is polynomial in the number of nodes n.

Conclusion

Since the problem can be solved in polynomial time, it lies in the complexity class **P**