1 $\mathcal{NP}$-completeness

To prove $\mathcal{NP}$-completeness, we look at the problem PartitionByPairs (PBP).

1.1 Transformation

Given an instance X of PBP, we do the following transformation ($T(X)$). We calculate the value $B = \frac{\sum_{i=1}^{2n} s_i}{2}$. For each pair (s_{2i-1}, s_{2i}) in S, where $i \in \{1, \ldots, n\}$, we construct a graph as show in Fig. 1, where the labels are the weights of the corresponding edge.

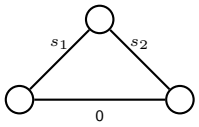

Figure 1: Transformation of a single pair

We order the set of edges, so the mirror of the edge with weight s_{2i-1} is s_{2i} and the mirror of an edge with weight 0 is an edge with weight 0 (if there is an odd number of pairs, the middle edge will be a zero that mirrors into itself). For multiple pairs, we chain multiple graphs together as shown in Fig. 2.

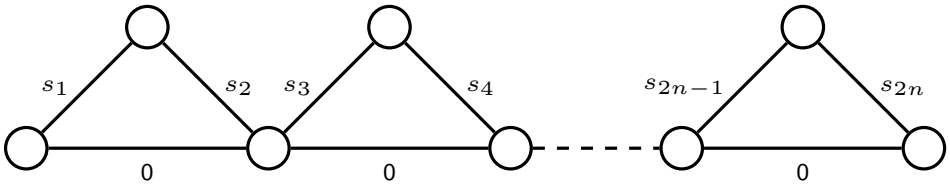

Figure 2: Transformation of several pairs

For example, the set $S = \{1, 2, 3, 4, 5, 6\}$ is transformed into a list of weights $W = [1, 3, 5, 0, 0, 0, 6, 4, 2]$, where the weight of edge i is the i'th element in W. Using this graph and the calculated value B we can query MFMST to answer PBP.

1.2 Proof

We do our transformation in polynomial time. The calculation of B is done in $O(n)$. For each pair, we construct a constant number of nodes and edges, so the graph can be created in $O(n)$, this means our transformation can be done in $O(n)$.

If the answer to the original problem instance X is YES, it means that a partition where we pick one from each pair equals B. It is possible to pick

a spanning tree where we pick one from each pair as shown in Fig 3. As the answer to the original problem was YES, and we can pick a spanning tree that uses an edge from each pair, the answer must also be YES to $T(X)$.

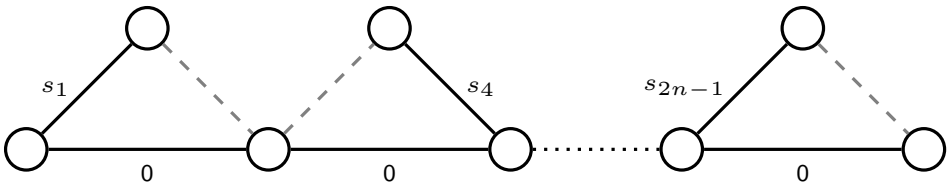

Figure 3: A spanning tree

We now assume the answer to the original problem is NO. In this case, we know that it is not possible to pick one edge from each pair in the transformation and get a value that is exactly B.

Any spanning tree must pick at least one from each pair, but can pick two (ignoring the 0-edge) and all edge weights are non-negative.

If we pick more than one edge from a pair, we increase both the weight of the spanning tree and the mirror, as the two edges we pick are mirrors of each other, and therefore will be in both sums.

In summary, this means that if the original answer is NO, we cannot pick a spanning tree using one from each pair, where the maximum value of the tree and the mirror is exactly B. As we need to pick at least one from each pair to have a spanning tree, and picking more than one, always increases both values, we cannot get a spanning tree that will make MFMST answer NO.

As our transformation runs in polynomial time, the original problem is $\mathcal{NP} - complete$, our transformation preserves the original answer and MFMST is in $\mathcal{NP}$, we can conclude that MFMST is $\mathcal{NP} - complete$.

2 Optimization Algorithm Description

2.1 General algorithm

Our algorithm does an exhaustive search of spanning trees, while storing the tree (T) scoring lowest in the equation

$$\max \left\{ \sum_{e_i \in T} w(e_i), \sum_{e_i \in T} w(e_{m+1-i}) \right\}$$

The spanning trees are found by doing so-called cuts and contracts. This creates a recursive computation as follows

If no edge remain, stop the calculation. Otherwise, pick an edge and:

1. Create a copy of the graph, where the end-points of this edge is merged into one point and any edges between them are removed. Store which edge was contracted. In practise we do this by maintaining a set of edges which should be considered contracted for the current calculations.

2. Create a copy of the graph where the edge is removed, if this disconnects a portion of the graph, stop calculating otherwise calculate all spanning trees on the copy.

Every leaf of this computation contains a unique spanning tree. It can be found by collecting all stored edges from leaf to root.

2.2 Tricks

Since all edges have positive weights, we can maintain the current "value" of the calculation that has been created, i.e. the edges that have already been picked. If we maintain a best found solution, we can stop finding more trees once we reach this value, thereby hopefully cutting of branches from the computation tree.

To increase the number of branches we can cut off, we can try to guess a good edge to contract at every step, using one of two heuristics

1. The edge e_i with the lowest score of $\max\{w(e_i), w(e_{m+1-i})\}$.

2. The edge e_i with the lowest score of $w(e_i) + w(e_{m+1-i})$

3 Running time

If the input graph has n nodes and m edges, we have the following running times.

- We can calculate all the heuristic values for edges in $O(m)$
- We can sort the edges by their heuristic value in $O(m)$.
- For every inner node in the computation tree we do the following:
  - Perform a depth-first-search to see if an edge added to the contracted set, creates a loop. This can be done in $O(n+m)$.
  - Perform a depth-first-search to see if removing an edge splits the current graph in two components. This can be done in $O(m+n)$.
- The computation tree is binary, and every leaf in the tree corresponds to a spanning tree. Cayleys formula states that a complete graph has $O(n^{n-2})$ spanning trees.

- The path from a leaf to the root of the tree is at most m.
- The computational tree has in the worst case, at most $O(m \cdot n^{n-2})$ nodes.

Summing this up, we get a running time of $O((m+n) \cdot m \cdot n^{n-2})$, as we perform two depth-first-searches for every inner node in the computation tree.