Acrobat: solution

It is easy to see that our job is to make our graph a connected Eulerian graph.

First we can fix the parities of the assistants on the first hill. To do this, note that the initial bipartite graph corresponds to a directed graph with n vertices and m edges and a change of type 1 is essentially a reversal of a directed edge. Making all degrees of part A even corresponds to making all out-degrees even and making all degrees of part B even corresponds to making all in-degrees even. To make all out-degrees even, we group the vertices with odd out-degree in pairs, and for each pair find a path between these two vertices (ignoring the direction) and reverse all edges on this path. This will change the parity of just the two endpoints. Furthermore, we can pick all paths from the same spanning tree, so we have made at most $n-1$ changes. We can make our algorithm run in linear time by noting that after having picked a rooted spanning tree (again, ignoring direction), an edge should be reversed iff the number of "bad" nodes in its subtree is odd. We can do this with a simple dfs. An other way to do this is to consider vertices in the order of an euler-tour traversal of some spanning tree, and pair up "bad" vertices that are adjacent in the ordering. So the total number of edges traversed/reversed will be again at most $n-1$.

The other part is easy, since we can just pair up "bad" assistants of the second hill and introduce a walk between them ($\leq n/2$ changes). Finally, we also have to make our graph connected. It is very easy to do this, for example by introducing a cycle (to avoid breaking parities) between representatives of different components. This will add at most $n-1$ changes.

So the total number of changes is $\leq 5n/2$. Also note that the conditions can be satisfied iff in the initial directed graph (but ignoring direction) every connected component has an even number of edges.