U2T is NP-complete

Dömötör Pálvölgyi *

Abstract

We show that U2T (the problem of deciding whether the edge set of a simple graph can be partitioned into two trees or not) is NP-complete.

One can easily see that whether a simple graph is a tree or not is in P. It was shown by Király that the problem of deciding if a simple graph is the disjoint union of three trees is NP-complete. Now we prove that for two trees the problem is also NP-complete. We will denote this problem by U2T. It obvious that U2T belongs to NP. To prove its completeness, we will show that Not-All-Equal SAT is reducible to U2T.

The Not-All-Equal SAT problem is the following: We are given polynomially many clauses over the variables $x_1, \dots, x_n$ and we have to decide whether there is an evaluation of the variables such that each clause contains both a true and a false literal. This is called a *good* evaluation. Eg., if the formula contains at least one clause of size one (like x_1), it does *not* have a *good* evaluation. This problem is well known to be NP-complete [1]. (Note that instead of this we could use SAT but then the maximum degree of our graph would significantly increase.)

Now we will construct a graph G from a given clause set $\mathcal{C}$. The graph G will consist of two main parts: L_i and C_j type subgraphs. A subgraph L_i belongs to each variable, while a subgraph C_j belongs to each clause from $\mathcal{C}$. Beside the L_is belonging to the variables, we also have two extra subgraphs of this type: L_0 and L_{n+1}. The vertex sets of the clauses are disjoint, while $V(L_i) \cap V(L_j) \neq \emptyset$ iff $|i-j| = 1$. If $j = i+1$ then $V(L_i) \cap V(L_j)$ is a single vertex denoted by t_i.

A variable component L_i consists of four vertices that form a cycle in the following order: t_{i-1}, v_i, t_i and $\overline{v}_i$. There are no edges inside the cycle. We would like to achieve that one of the trees contains the edges from t_{i-1} through v_i to t_i, while the other from t_{i-1} through $\overline{v}_i$ to t_i. For simplicity, we denote t_{-1} by α and t_{n+1} by ω. Both trees are desired to trail from α to ω. The idea is that we want to force one of the

*Dept of Comp. Sci., ELTE, Budapest
Computer Networks Lab, ELTE, Budapest

trees to go through those v_is for which x_i is true.

Before we start the construction of the clause components, we introduce a notation: We say that two vertices u and w are linked with a *purple* edge if
(1) There is no edge between u and w.
(2) The smaller connectivity component of $G \setminus \{u, w\}$ (called *purple component*) consists of four vertices: $v_1^{uw}, v_2^{uw}, v_3^{uw}$ and v_4^{uw}.
(3) The v_i^{uw} vertices form a cycle in this order.
(4) The vertices u and w are connected only to v_1^{uw}.
(*See Figure 1.*)
This is a very useful structure because if G is the union of two trees, then they both have to enter this purple component since a tree cannot contain a cycle. So if the vertices are linked with a purple edge and $E(G) = E(T) \dot{\cup} E(F)$ (where T and F denote the two trees), then it means that $u \in V(T)$ and $w \in V(F)$ or $u \in V(F)$ and $w \in V(T)$ (or maybe both).

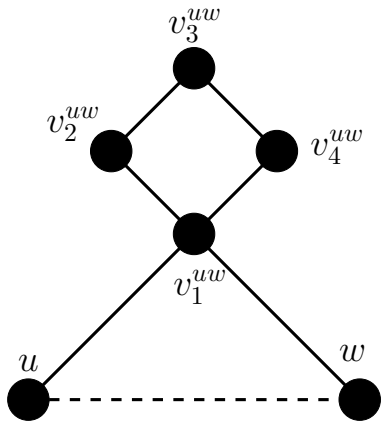

Figure 1. A *purple* edge.

A clause component C_j consists of $3k$ vertices where k is the size of the jth clause whose literals are denoted by $l_1^j, \ldots, l_k^j$. A cycle of length $2k$ is formed by the following vertices in this order: $p_1^j, q_1^j, p_2^j, q_2^j, \ldots p_k^j, q_k^j$. The other k vertices are denoted by $r_1^j, \ldots r_k^j$. The vertex r_i^j is always connected to p_i^j and it is also connected to v_m if l_i^j is x_m or to $\overline{v}_m$ if l_i^j is $\overline{x}_m$. Furthermore, there is a purple edge between r_i^j and q_i^j. This will ensure that a tree coming in to the *clause circle* through a r_i^j, cannot „go out". (*See Figure 2. for a graph with one clause. The dashed edges mean purple edges.*)

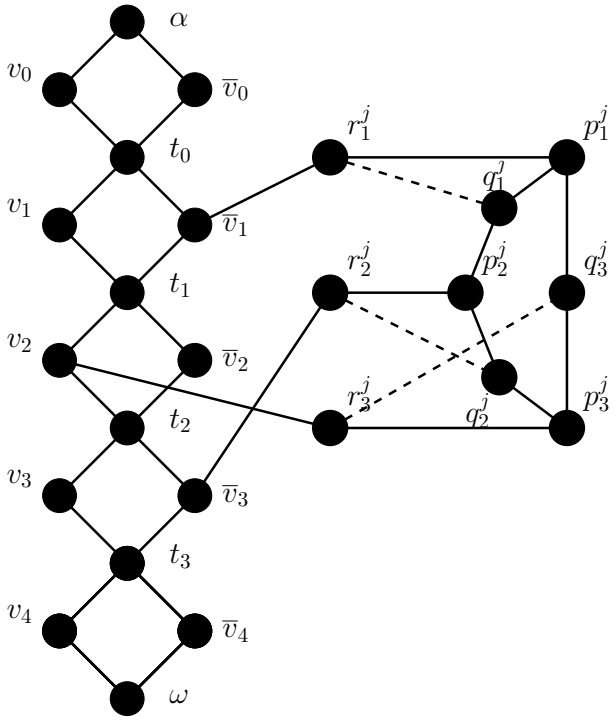

Figure 2. The graph with the jth clause what is $\overline{x}_1, x_2, \overline{x}_3$.

We are ready with the construction, now we have to prove that it is correct. The easier part is to show that if our NOT-ALL-EQUAL SAT problem has a good evaluation, then we have two trees, T and F. First let us fix a good evaluation. Let the tree T contain the path from α to ω through the v_is for true x_is and through the $\overline{v}_i$s for false x_is. Similarly F trails from α to ω through the $\overline{v}_i$s for true x_is and through the v_is for false x_is. If a tree contains v_i (or $\overline{v}_i$), let it also contain the path to the proper p_m^j if x_i (or $\overline{x}_i$) is in the jth clause. This way both trees enter each *clause circle* since the evaluation satisfied our NOT-ALL-EQUAL SAT problem. Let the two edges from p_i^j to p_{i+1}^j belong to the tree that does *not* contain r_i^j. This guarantees that we have no problem with the purple edges and one can easily see that the trees remain connected. So we are done with this part.

To prove the other part, let us suppose that $G = T \dot{\cup} F$. We know that v_0 or $\overline{v}_0 \in V(T)$, otherwise F would contain the whole L_0 component and thus have a cycle. We can suppose $\alpha \in V(T)$. We can also suppose $\omega \in V(F)$. Let us direct all the edges of the trees away from here. Similarly, we can suppose $\omega \in V(T)$ and $V(F)$. Now some basic observations.

Proposition 1. *There are no edges coming out of the purple components.*

Proof. Both trees have to enter each purple component since a tree cannot contain a cycle and since there are only two edges connecting a purple component to the rest of the graph, both of them must be directed toward the purple component. □

This means that the trees cannot go through purple edges.

Proposition 2. *There are no edges coming out of the clause components.*

Proof. Let us suppose that the edge from r_i^j going to some v_m (or $\overline{v}_m$) is directed away from r_i^j and is in T. This implies $p_i^j r_i^j \in T$ as well because T cannot enter r_i^j through the purple edge. But because $r_i^j \notin V(F)$, therefore $q_{i+1}^j \in V(F)$ since they are linked with a purple edge, so T must have entered p_i^j from the direction of p_{i-1}^j through q_{i-1}^j. But then $q_{i-1}^j \notin V(F)$, so $r_{i-1}^j \in V(F)$. This means T entered p_{i-1}^j from the direction of p_{i-2}^j. And we can go on so until we get back to p_i^j, what gives a contradiction. □

So now we know that the clauses are dead ends as well as the purple components. Since T trails from α to ω, it must contain v_i or $\overline{v}_i$ for each i. Similarly v_i or $\overline{v}_i \in V(F)$. Moreover, for $1 \leq i \leq n$, no v_i or $\overline{v}_i$ can be contained in both because it would destroy the connectivity of one of the trees. So we can define x_i to be true if $v_i \in V(T)$. Now the only thing left to show is that the literals in the clauses are not equal. But if they were, then the C_j component of the clause would be connected to only one of the trees and hence that tree would contain a circle, contradiction. So we have shown that each tree partition yields a proper evaluation. □

Now we give an estimation on the maximum degree of the graph that we constructed. Clearly all the vertices, except the v_is and $\overline{v}_i$s, can have at most four edges. A v_i (or $\overline{v}_i$) has degree equal to two plus the occurances of the literal it belongs to. But a Not-All-Equal SAT problem is easily reducible to a Not-All-Equal SAT-(2;2) problem (meaning that each literal can occur at most twice). If a literal l would occur in at least three clauses, then let us execute the following operation until we have at most two of each literal: Replace $(C_1, l), (C_2, l), (C_3, l)$ with $(l, \overline{z}), (C_1, l), (C_2, z), (C_3, z)$ where z is a new variable. Therefore the decision of whether a simple graph is the disjoint union of two trees or not is NP-complete even for graphs with maximum degree four.

References

[1] M. Garey and D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, New York, 1979.