SAT and MaxSAT Encodings for Trees Applied to the Steiner Tree Problem

Ricardo Tavares de Oliveira and Fabiano Silva
Federal University of Paraná
POBox 19081, 81531-980, Curitiba, Brazil
Email: {rtoliveira,fabiano}@inf.ufpr.br

***Abstract*—In this paper we present some SAT and MaxSAT codifications for trees in graphs. We review two known encodings and suggest four new ones. We categorize the encodings into three categories: *Absolute* encodings state that each vertex must be in a fixed position in some structure; *Relative* encodings state the relative positions between the vertices in some structure; *Counting-based* encodings use theoretical graph properties to hardcode the degree of each vertex in a tree. We use these encodings to reduce the Steiner Tree Problem in Graphs to (Partial Weighted) MaxSAT and compare their efficiency to solve random instances and ones from the SteinLib benchmark. The experiments strongly suggest that relative encodings are more efficient than absolute ones, but there is not an overall best encoding between relative and counting-based ones.**

I. Introduction

Due to the growing efficiency of SAT and MaxSAT solvers, the interest on solving NP-Complete and some NP-Hard problems with SAT or MaxSAT has grown. Among these problems, the ones related to graphs, like the Hamiltonian Cycle, Steiner Tree, Isomorphism and others problems, are widely studied.

In reductions from graph problems to SAT or MaxSAT, it is usual to encode structures like paths, cycles, independent sets, vertex mapping, and others [1], [2], [3], [4], [5], [6], [7]. Each encoding can be classified according to the semantic of its variables [8], [7].

In this paper, we present encodings for *trees*. Given a graph G, we build some SAT or MaxSAT instances whose solutions represent a tree in G. We review two known encodings and show four new ones, and classify them.

Our main application is the Steiner Tree Problem: Given a weighted graph $G = (V, E, w : E \rightarrow \mathtt{N})$ and a collection of its vertices $S \subseteq V$ (called the *terminal* ones), this problem consists of finding a tree that contains all terminals vertices whose sum of the weights of its edges is minimal. This problem is known to be NP-Hard [9], and has applications, for instance, in Computation Geometry and Circuit Design [10]. As we show, the problem can be polynomially reduced to MaxSAT through propositional formulae using tree encodings.

This paper is organized as follows: Section II provides preliminary definitions and notations. Section III provides some background on known encodings for graph problems. Section IV describes absolute encodings for trees and for the Steiner Tree Problem. Section V describes relative encodings for the same problem, while section VI describes counting-based encodings. Finally, section VII shows some experimental results, while section VIII concludes and suggest future works.

II. Preliminaries

Let us recall some SAT and MaxSAT definitions. A *boolean variable* is a variable that may assume a truth value. A *literal* is a variable x_i or its negation, $\neg x_i$. A *clause* C is a disjunction of literals. A clause $C = (\neg l_1 \vee ... \vee \neg l_n \vee p_1 \vee ... \vee p_m)$ can also be written as $C = (l_1 \wedge ... \wedge l_n) \rightarrow (p_1 \vee ... \vee p_m)$ when convenient. An *assignment* is a set of literals where x_i and $\neg x_i$ do not occur simultaneously, for any variable x_i.

A clause is *satisfied* by an assignment if there is a literal occurring both in the clause and the assignment. A Conjunctive Normal Form (CNF) formula is a conjunction of clauses. An assignment satisfies a CNF formula if it satisfies all its clauses. An assignment that satisfies a CNF formula is a *model* of it.

The Satisfability problem (SAT) consists in deciding whether a given CNF formula has a model. The Partial Weighted Maximum Satisfability Problem (MaxSAT) consists in, given a set of *hard* clauses whose conjunction forms the *hard* formula and a set of weighted clauses (called the *soft* clauses, denoted by $(C, W(C))$), finding a model of the *hard* formula whose sum of the weights of the satisfied *soft* clauses is maximized.

We denote by V the set of vertices of a graph, by E the set of its edges if it is undirected, and by A the set of its arcs if it is directed. We denote by $w(e_i)$, the weight of the edge e_i. Also, we denote the set of neighbors of a given vertex v_i by $N(v_i) = \{v_{i^1}, v_{i^2}, ..., v_{i^{|N(v_i)|}}\}$. A vertex is not considered a neighbor of itself.

III. Background

In this section we present some background on encodings for graph problems. Firstly, Prestwich suggested some encodings for the Hamiltonian Cycle Problem [8]. In his encodings, Prestwich suggested to create a boolean formula that, given a graph G, is satisfiable iff G contains a Hamiltonian Cycle. This formula is created by encoding some structure relative to the vertices in G and constraints applied to it. In this case, a permutation of the vertices is encoded and constraints are added to ensure that there is an edge in the graph between each pair of adjacent vertices in the permutation.

Prestwich suggested two encodings: the *absolute* encoding and the *relative* encoding. In the absolute encoding, all the boolean variables used in the formula encode the *exact* ("absolute") position of all the vertices in the permutation. Clauses ensure that: each vertex occurs in the permutation; no vertex occurs more than once in the permutation; no two

vertices occurs at the same position of the permutation; the vertex chosen to be the first one occurs at the first position of the permutation; and there are no pair of adjacent vertices not connected by a single edge in G.

Absolute encodings can also be classified according to the semantic of the models of the formula, as detailed by Velev&Gao [7]. In the *direct* encoding, $|V|^2$ variables are created: variable $x_{i,j}$ is true iff $v_i \in V$ occurs at the position j of the permutation. In the *log* encoding, $|V|\lceil \lg |V| \rceil$ variables are created, and each variable encodes a bit in the binary representation of an index in the permutation. *muldirect*, *ITE-linear* and *ITE-log* absolute encodings can also be cited, among others [7]. In this paper, we consider the direct encoding to build the absolute codifications.

In the relative encoding, the boolean variables used in the formula encode the *relative* order between vertices in the permutation. For each pair of vertices in G, two types of variables are defined: variable $s_{i,j}$ states that v_j is the *successor* of v_i in the permutation, and the variable $o_{i,j}$ states that v_i *precedes* v_j in it. Clauses are then added to the formula to ensure that every vertex occurs exactly once in the permutation and has exactly one successor and one antecessor, and the induced sequence does not contain any pair of adjacent vertices where there is no edge between them in the graph.

Velev&Gao noticed that $s_{i,j}$ can be defined only when the edge $\{v_i, v_j\}$ is in the graph. Also, since $o_{i,j} \leftrightarrow \neg o_{j,i}$ holds in any model of the formula, then $o_{i,j}$ can be defined only for pairs of vertices where $i < j$, and all literals $o_{i,j}$, with $i > j$, can be replaced by $\neg o_{j,i}$. This makes the formula smaller and also easier to be solved by a SAT solver [6].

The ideas behind these encondigs can be adapted to solve problems by SAT other than Hamiltonian Cycle. For instance, a reduction from the (decision) Treewidth Problem to SAT is suggested by Samer&Veith [5]. Their reduction uses an encoding that can be classified as relative, since their encoding uses boolean variable to state whether a vertex succeeds and precedes another in a linear ordering, analogous to the variables $s_{i,j}$ and $o_{i,j}$ described previously. Like Velev&Gao, they also replace $o_{i,j}$ by $\neg o_{j,i}$ when $i > j$.

Also, reductions from Graph $k-$Coloring that use absolute encodings and its variants are presented by Van Gelder [11]. He shows a direct encoding, where a variable $x_{i,j}$ states that vertex $v_i \in V$ is colored with color j, where $1 \leq j \leq k$. A log encoding is also shown, where the binary representation of the index of the color of each vertex is encoded.

In this paper, we study some known encodings for trees and suggest new ones, and use them to solve the Steiner Tree Problem. In our previous work, we suggested an encoding for this problem that we do not consider absolute nor relative. This encoding - which we classify as *counting-based*, use cardinality operators and theoretical graph properties to encode the existence of paths between vertices [1]. In particular, we encoded paths between vertices using a class of cardinality operators, and then presented a way to use these encodings to obtain a tree. This construction is reviewed in this paper.

In the next sections we describe two known and four new encodings for trees and the Steiner Tree Problem. Two of them are considered absolute. Other two of them are considered relative, and the remaining two encodings are considered counting-based.

IV. Absolute encodings

In this section we present two absolute MaxSAT encodings that ensure the existence of a tree in the given graph. Given a graph G and a set of its vertices $S \subseteq V$, both encodings build a *hard* formula whose models describe a *connected subgraph* of G containing the vertices in S. *Soft* clauses are then added to make this subgraph minimal, resulting in a tree. For the Steiner Tree problem, the set S consists of all terminal vertices. However, one can set $S = V$, for instance, to encode a tree that contains all vertices in the graph.

We present two absolute encodings, which we call *DFT-based* and *Path-based*. The first one is based on an idea given by Kautz et al. [3], while the other one is a new encoding.

A. DFT-based encoding

Kautz et al. suggested an idea for a reduction from the Steiner Tree Problem to MaxSAT where each optimal model of the resulting formula describes a depth-first traversal (DFT) of a tree in the graph [3].

As described, given a weighted graph G and a subset of its vertices $S \subseteq V$, this encoding generates a *hard* formula $F_{DFT}(G, S)$ whose models describe a connected subgraph that contains the vertices in S. *Soft* clauses are then used to minimize such subgraph, which results in a tree.

Firstly, let us make the graph *directed*: let us replace each edge $\{v_i, v_j\} \in E$ with two *arcs* $(v_i, v_j), (v_j, v_i) \in A$.

Any subgraph of G can be described by the cyclical sequence of its arcs in the order they would be visited in a depth-first traversal. Also, if we allow some arcs to appear more than once in the sequence, we can fix its size to the largest possible, $2|E|$.

As an example, consider that $G' = (\{v_1, v_2, v_3, v_4\}, \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\})$ is a subgraph of a graph G that contains 4 edges. A possible sequence of $2 \times 4 = 8$ arcs that describes this subgraph is $((v_2, v_1), (v_1, v_3), (v_3, v_1), (v_1, v_4), (v_4, v_1), (v_1, v_2), (v_2, v_1), (v_1, v_2))$.

For each arc $(v_i, v_j) \in A$, and for each $1 \leq t \leq 2|E|$, let us associate a boolean variable $x_{i,j,t}$ that states that the arc (v_i, v_j) occurs at position t in the sequence. To ensure that the sequence describes a valid traversal, we suggest to build $F_{DFT}(G, S)$ with the following clauses:

a. For each $(v_i, v_j) \in A$ and for each $1 \leq t \leq 2|E|$, $x_{i,j,t} \rightarrow (x_{j,j^1,t'} \vee x_{j,j^2,t'} \vee ... x_{j,j^{|N(v_j)|},t'})$, where $t' = (t \ mod \ 2|E|) + 1$. These $4|E|^2$ clauses state that, if the arc (v_i, v_j) occurs at position t , then some arc from v_j must follow it in the next position, t'.

b. For each $(v_i, v_j) \in A$ and for each $1 \leq t \leq 2|E|$, $x_{i,j,t} \rightarrow (x_{j,i,1} \vee x_{j,i,2} \vee ... \vee x_{j,i,2|E|})$. These $4|E|^2$ clauses state that, if the arc (v_i, v_j) occurs in the sequence at all, then its inverse arc must occur, too. Hence, each edge of the original graph must be visited in both ways or must not be visited at all during the transversal.

c. For each $1 \leq t \leq 2|E|$ and for each pair of distinct arcs $(v_i, v_j), (v_k, v_l) \in A$, $x_{i,j,t} \rightarrow \neg x_{k,l,t}$. These $8|E|^3 - 4|E|^2$ clauses state that there is at most one arc in each position of the sequence.

d. For each $v_s \in S$, and for each $1 \leq t \leq 2|E|$, $x_{s,s^1,t} \vee x_{s,s^2,t} \vee ... \vee x_{s,s^{|N(v_s)|},t}$. These $2|S||E|$ clauses state that some arc from v_s occours in the sequence, for all $v_s \in S$.

The formula $F_{DFT}(G, S)$ is given by the conjunction of all the clauses described above. Every model of this formula induces a connected subgraph of G containing the vertices in S. Since the smallest such subgraph is a (Steiner) tree, we can reduce the Steiner Tree Problem to MaxSAT by defining *soft* clauses that ensure the subgraph is minimized.

Kautz et al. suggested creating a boolean variable $y_{i,j}$ for each edge $\{v_i, v_j\} \in E$ and adding the *soft* clause $(\neg y_i, w(\{v_i, v_j\}))$ to the formula. The variable $y_{i,j}$ states the presence of (v_i, v_j) or (v_j, v_i) in the sequence. The relationship between this variables and the ones previously defined may be given by *hard* clauses, to be added to $F_{DFT}(G, S)$, that describe both ways of the following relationship: $y_{i,j} \leftrightarrow (x_{i,j,1} \vee ... \vee x_{i,j,2|E|} \vee x_{j,i,1} \vee ... \vee x_{j,i,2|E|})$.

This encoding is classified as absolute because its variables encode the exact positions the arcs occur in the sequence. It is not straightforward to adapt this encoding to a relative one, since an arc may appear more than once in the sequence, and thus the position of its "next arc" may not be unique.

B. Absolute Path-based encoding

The DFT-based encoding codifies the existence of a minimal connected subgraph of the given graph. In our previous work, we suggested to encode the same constraint by ensuring the existence of paths between pair of vertices [1]. The Path-based encoding is a new way to encode such existence.

First, given a graph G and two of its vertices $v_a, v_b \in V$, let us create a formula $F_{APB}(G, v_a, v_b)$ that is satisfiable iff there is a path between v_a and v_b in G. This formula is used as a subformula in the Path-based encoding.

A path from v_a to v_b can be described with a *sequence* of some of the vertices in the graph ($v_1 = v_a$, v_2, v_3, ..., v_{k-1}, $v_k = v_b$), where k is the number of vertices in the path. There must be an edge between v_i and v_{i+1} for all $1 \leq i < k$. Only the vertices present in the described path must occur in the sequence. In the worst case, however, the path contains all vertices, and the sequence has then all $|V|$ vertices.

For each vertex $v_i \in V$ and for each position $t, 1 \leq t \leq |V|$, let us associate a boolean variable $x_{i,t}$ that states that v_i occurs at the position t of the sequence.

$F_{APB}(G, v_a, v_b)$ consists of the following clauses:

a. $x_{a,1}$. This unit clause ensures that v_a occurs at the first position of the sequence.

b. $x_{b,2} \vee x_{b,3} \vee ... \vee x_{b,|V|}$. This clause ensures that x_b occurs in the sequence at some position.

c. For each vertex $v_i \in V, v_i \neq v_b$ and for each $1 \leq t < |V|$, $x_{i,t} \rightarrow (x_{i^1,t+1} \vee x_{i^2,t+1} \vee ... \vee x_{i^{|N(v_i)|},t+1})$. These $|V|^2 - |V|$ clauses ensure that, if v_i occurs at some position t in the sequence and it is not v_b, then one of its neighbors follows it at position $t + 1$. This ensure that there is an edge between adjacent vertices in the sequence, and thus the sequence describes a valid path.

d. For each $1 \leq t \leq |V|$ and for each pair of distinct vertices $v_i, v_j \in V$, $x_{i,t} \rightarrow \neg x_{j,t}$. These $|V|^3 - |V|^2$ clauses ensure that there is at most one vertex in each position of the sequence.

e. For each vertex $v_i \in V$ and for each pair of positions $1 \leq t < t' \leq |V|$, $x_{i,t} \rightarrow \neg x_{i,t'}$. These $O(|V|^3)$ clauses ensure that all vertices occur in the sequence at most once.

These clauses are sufficient to ensure that the sequence describes a path from v_a to v_b in the given graph. A vertex $v_i \in V$ is present in the described path iff it occurs at some position in the sequence, ie, iff $x_{i,t}$ is true for any $1 \leq t \leq |V|$.

For each edge $\{v_i, v_j\} \in E$, let us associate two boolean variables $z_{i,j}$ ($z_{j,i}$) that state that v_j (v_i) is the successor of v_i (v_j) in the sequence. These variables are true iff that edge is present in the path being described.

Their relationship with the previously defined variables are given by the following relation, to be converted to clauses to be added to the formula: $z_{i,j} \leftrightarrow (x_{i,1} \wedge x_{j,2}) \vee (x_{i,2} \wedge x_{j,3}) \vee ... \vee (x_{i,|V|-1} \wedge x_{j,|V|})$. The relationship for variable $z_{i,j}$ is analogous.

Once the formula $F_{APB}(G, v_a, v_b)$ is defined, we can use it as a subformula to create a formula $F_{APB}(G, S)$ whose each model describes a connected subgraph of G containing the vertices in S.

This construction is similar to the one suggested in our previous work [1]. Let us define $S' = (v_{s1}, v_{s2}, ..., v_{s|S|})$ as an arbritrary permutation of S. A subgraph containing all vertices in S is connected iff it contains a path between each two adjacent vertices in S', ie, between v_{sg} and $v_{s(g+1)}$, for all $1 \leq g < |S|$.

Let us consider $|S| - 1$ graphs $G_1, ..., G_{|S|-1}$ which are all identical to the original graph G. For each graph $G_g, 1 \leq g < |S|$, consider the formula $F_{APB}(G_g, v_{sg}, v_{s(g+1)})$, that states that there is a path between v_{sg} and $v_{s(g+1)}$ in graph G_g. These formulae contain the boolean variables $x_{i,t,g}$ ($z_{i,j,g}$), whose meaning is identical to $x_{i,t}$ ($z_{i,j}$), w.r.t. G_g.

The formula $F_{APB}(G, S)$ is defined simply as the conjunction of these formulae, since the union of the paths described by the models of these formulae results in a connected subgraph containing all vertices in S.

Soft clauses are then used to minimize the described subgraph. For each edge $\{v_i, v_j\} \in E$, let us associate a boolean variable $y_{i,j}$ that states that this edge is present in the subgraph being described. The relationship between this variables is encoded by *hard* clauses translating both ways of the following relation, to be added to $F_{APB}(G, S)$: $y_{i,j} \leftrightarrow (z_{i,j,1} \vee z_{j,i,1} \vee z_{i,j,2} \vee z_{j,i,2} \vee ... z_{i,j,|S|-1} \vee z_{j,i,|S|-1})$. Like the DFT-based encoding, let us add an unit *soft* clause $(\neg y_{i,j}, w(\{v_i, v_j\}))$ for each edge $\{v_i, v_j\} \in E$. These clauses ensures the subgraph being described is minimal and thus is a (Steiner) tree.

A relative version of this encoding is described in the next section.

V. Relative encodings

In this section we present relative encodings of trees in graphs. We suggest a relative adaptation of the Path-based encoding described in the previous section, and a new relative encoding, which we call *Parental-based encoding*. These encodings are described next.

A. Relative Path-based encoding

The Absolute Path-based encoding can be adapted to a relative one as Prestwich did to Hamiltonian paths [8].

Like the absolute encoding, let us build a formula $F_{RPB}(G, v_a, v_b)$ that is satisfiable iff there is a path between v_a and v_b in G, and define $F_{RPB}(G, S)$ as the conjunction of $|S| - 1$ such formulae.

For each one of them, let us consider a sequence of vertices $(v_1 = v_a,\ v_2,\ v_3,\ ...,\ v_{k-1},\ v_k = v_b)$ that describes a path between v_a and v_b, as shown in the previous section.

For each edge $\{v_i, v_j\} \in E$, let us associate two boolean variables $s_{i,j}$ ($s_{j,i}$) that state that v_j (v_i) is the *successor* of v_i (v_j) in the sequence. Also, for each pair of distinct vertices $v_i, v_j \in V$, let us associate a boolean variable $o_{i,j}$ that states that v_i *precedes* v_j in the sequence.

We build $F_{RPB}(G, v_a, v_b)$ with the following clauses:

a. $s_{a,a^1} \vee s_{a,a^2} \vee ... \vee s_{a,a^{|N(v_a)|}}$. This clause ensures that v_a has a successor in the sequence.

b. For each vertex $v_i \in V$ and for each pair of distinct vertices $v_j, v_k \in N(v_i)$, $s_{i,j} \rightarrow \neg s_{i,k}$, and $s_{j,i} \rightarrow \neg s_{k,i}$. These $O(|V|^3)$ clauses ensure that no vertex has more than one successor or one antecessor in the sequence.

c. For each vertex $v_i \in V$ with $v_i \neq v_b$, and for each vertex $v_j \in N(v_i)$, $s_{j,i} \rightarrow (s_{i,i^1} \vee s_{i,i^2} \vee ... \vee s_{i,i^{|N(v_i)|}})$. These $O(|E|)$ clauses ensure that, if a vertex v_i has some antecessor v_j in the sequence, then it has a successor in it too, except when $v_i = v_b$.

d. For each triple of distinct vertices $v_i, v_j, v_k \in V$, $(o_{i,j} \wedge o_{j,k}) \rightarrow o_{i,k}$. These $O(|V|^3)$ clauses encode the transitivity of the preceding relationship.

e. For each pair of distinct vertices $v_i, v_j \in V$, $o_{i,j} \rightarrow \neg o_{j,i}$. These $|V|^2 - |V|$ clauses encode the antisymmetry of the preceding relationship.

f. $o_{a,b}$. This unit clause ensures that v_a precedes v_b in the sequence.

g. For each pair of distinct vertices $v_i, v_j \in V$, $s_{i,j} \rightarrow o_{i,j}$. These $|V|^2 - |V|$ clauses encode the natural relationship between both types of variables.

These clauses are sufficient to encode a path between v_a and v_b in G. A vertex $v_i \in V$ is present in the described path iff it has a successor or an antecessor in the sequence, ie, if $s_{i,j}$ or $s_{j,i}$ is true for any $v_j \in N(v_i)$.

It is worth noticing that, unlike Velev&Gao's encoding, we cannot replace the variable $o_{i,j}$ by $\neg o_{j,i}$ when $i > j$, since there may be vertices that are not present in the sequence at all, and thus the relation between them is undefined. Hence, $o_{i,j}$ and $o_{j,i}$ can both be false if v_i and v_j are not in the path.

Unlike the absolute version of this encoding, we do not need to create a boolean variable $z_{i,j}$ for each $\{v_i, v_j\} \in E$ stating the fact that v_j precedes v_i in the sequence, since this statement is given by the variable $s_{i,j}$ already.

Once the formula $F_{RPB}(G, v_a, v_b)$ is defined, we can again use it as a subformula to create a formula $F_{RPB}(G, S)$ whose models describe a connected subgraph of G containing the vertices in S. This formula can be built with an arbitrary permutation S' of the elements in S, as described previously. Again, we create a boolean variable $y_{i,j}$ for each edge $\{v_i, v_j\} \in E$ and add clauses to $F_{RPB}(G, S)$ that describe the relation $y_{i,j} \leftrightarrow (s_{i,j,1} \vee s_{j,i,1} \vee s_{i,j,2} \vee s_{j,i,2} \vee ... s_{i,j,|S|-1} \vee s_{j,i,|S|-1})$, as done in the Absolute Path-based encoding.

Finally, an unit *soft* clause $(\neg y_{i,j}, w(\{v_i, v_j\}))$ is added to the formula for each $\{v_i, v_j\} \in E$, as done in all presented encodings.

B. Parental-based encoding

All previously described encodings generate a formula whose models describe a subgraph of a given graph. It becomes a tree through MaxSAT minimization. We describe a new encoding that encodes a tree directly with the *hard* formula.

Let us build a formula $F_{PrB}(G, S)$ whose models describe a tree containing the vertices of S. This tree is rooted in an arbritrary vertex $v_r \in S$.

For each edge $\{v_i, v_j\} \in E$, let us associate two boolean variables $p_{i,j}$ ($p_{j,i}$) that state that v_j (v_i) is the *parent* of v_i (v_j) in the tree. Also, for each pair of distinct vertices $v_i, v_j \in V$, let us associate a boolean variable $a_{i,j}$ that states that v_j is an *ancestor* of v_i in the tree. We build $F_{PrB}(G, S)$ with the following clauses:

a. For each vertex $v_s \in S$ with $v_s \neq v_r$, $p_{s,s^1} \vee p_{s,s^2} \vee ... \vee p_{s,s^{|N(v_s)|}}$. These $|S| - 1$ clauses state that every vertex in S, except for the root, must have a parent in the tree;

b. For each vertex $v_i \in V$ with $v_i \neq v_r$ and for each pair of distinct vertices $v_j, v_k \in N(v_i)$, $p_{i,j} \rightarrow \neg p_{i,k}$. These $O(|V|^3)$ clauses state that no vertex has more than one parent.

c. For each vertex $v_i \in V$ with $v_i \neq v_r$ and and for each vertex $v_j \in N(v_i)$ with $v_j \neq v_r$, $p_{j,i} \rightarrow (p_{i,i^1} \vee p_{i,i^2} \vee ... \vee p_{i,i^{|N(v_i)|}})$. These $O(|E|)$ clauses state that, if the vertex v_i is not the root and has some child v_j in the tree, then it must also have a parent in it.

d. For each vertex $v_i \in V$ with $v_i \neq v_r$ and for each vertex $v_j \in N(v_i)$, $p_{i,j} \rightarrow a_{i,j}$. These $O(|E|)$ clauses state the direct relationship between the parent and ancestor relations.

e. For each triple of distinct vertices $v_i, v_j, v_k \in V$, $(a_{i,j} \wedge a_{j,k}) \rightarrow a_{i,k}$. These $O(|V|^3)$ clauses encode the transitivity relationship of the ancestor relation.

f. For each pair of vertices $v_i, v_j \in V$, $a_{i,j} \rightarrow \neg a_{j,i}$. These $|V|^2 - |V|$ clauses encode the antisymmetric relationship of the ancestor relation.

g. For each vertex $v_s \in S, v_s \neq v_r$, $a_{s,r}$. These $|S|-1$ unit clauses encode the fact that v_r must be an ancestral of all the other vertices in S.

Given a model of this formula, a vertex v_i is present in the tree described by it iff it is the root or it has some parent assigned to it, ie, $p_{i,j}$ is true, for some $v_j \in N(v_i)$.

We also create a boolean variable $y_{i,j}$ for each edge $\{v_i, v_j\} \in E$ which is true iff that edge is in the described tree. This variable is related to the other ones simply by the relation $y_{i,j} \leftrightarrow (p_{i,j} \vee p_{j,i})$, where $p_{i,j}$ ($p_{j,i}$) is removed if v_i (v_j) is the root. As usual, there is a *soft* unit clause $(\neg y_{i,j}, w(\{v_i, v_j\})$ for each edge $\{v_i, v_j\} \in E$.

It is worth mentioning that, again, we cannot replace the variable $a_{i,j}$ by $\neg a_{j,i}$ when $i > j$, since the ancestor relation does not describe a total order of the vertices: there may be two vertices $v_i, v_j \in V$ where v_i is not an ancestor of v_j nor v_j is an ancestor of v_i.

VI. Counting-based encodings

Let us recall the idea suggested in our previous work to reduce the Steiner Tree Problem to MaxSAT by creating some subformulae that, given a graph G and two of its vertices $v_a, v_b \in V$, is satisfiable iff there is a path between v_a and v_b in G. The union of these paths results in a connected subgraph of G, to be minimized thought MaxSAT minimization [1].

A path between v_a and v_b in a graph G exists iff there is a subgraph of G in which the degree of both v_a and v_b is exactly 1, and the degree of all other vertices is 0 or 2 [1].

Observing this, we suggested to associate a boolean variable $x_{i,j}$ for each edge $\{v_i, v_j\} \in E$ and create the hard formula by using cardinality operators to count the degree of each vertex in the subgraph described by a model of the formula. Previously, we suggested to translate these cardinality operators to CNF by using recursive counters shown by Muller&Preparata [12]. In this paper, we call this known encoding the *Recursive-Counter-based* one.

These operators can also be translated Pseudo-Boolean (PB) constraints, which consists of $0-1$ linear programming equations. The constraints that encode a subgraph of G containing a path between v_a and v_b are:

a. $x_{a,a^1} + x_{a,a^2} + ... + x_{a,a^{|N(v_a)|}} = 1$ and $x_{b,b^1} + x_{b,b^2} + ... + x_{b,b^{|N(v_b)|}} = 1$. These constraints state that the degree of both v_a and v_b must be equal to 1.

b. For each $x_i \in V \setminus \{x_a, x_b\}$, $(x_{i,i^1} + x_{i,i^2} + ... + x_{i,i^{|N(v_i)|}} = 0) \vee (x_{i,i^1} + x_{i,i^2} + ... + x_{i,i^{|N(v_i)|}} = 2)$. These constraints state that the degree of all other vertices must be equal to 0 or 2.

We can rewrite these constraints to use only the $\leq$ operator. The constraint related to v_a can be rewritten as $(x_{a,a^1} + x_{a,a^2} + ... + x_{a,a^{|N(v_a)|}} \leq 1) \wedge (\neg x_{a,a^1} + \neg x_{a,a^2} + ... + \neg x_{a,a^{|N(v_a)|}} \leq |N(v_a)| - 1)$. The one related to v_b is analogous. In a similar way, every constraint of the form $x_{i,i^1} + x_{i,i^2} + ... + x_{i,i^{|N(v_i)|}} = 2$ can be rewritten as $(x_{i,i^1} + x_{i,i^2} + ... + x_{i,i^{|N(v_i)|}} \leq 2) \wedge (\neg x_{i,i^1} + \neg x_{i,i^2} + ... + \neg x_{i,i^{|N(v_i)|}} \leq |N(v_i)| - 2)$. Finally, the constraint $x_{i,i^1} + x_{i,i^2} + ... + x_{i,i^{|N(v_i)|}} = 0$ can simply be replaced by $x_{i,i^1} + x_{i,i^2} + ... + x_{i,i^{|N(v_i)|}} \leq 0$.

Instead of using counters, we now suggest translating these PB constraints to CNF by creating a Reduced Ordered Binary Decision Diagram (ROBDD) for each one, which is a canonical representation of a boolean function. We call this encoding the *ROBDD-based* one. For a background on ROBDDs, the reader may refer to [13].

For each constraint, we create an equivalent ROBDD as suggested by Abio et al. [13]: Given a total order of the variables present in the constraint, we generate an Ordered BDD whose root node represents the constraint itself, and the two child of a node represent the propagation of the respective truth value of the variable in its level in the constraint. We then remove irrelevant nodes and similar subgraphs, generating an unique ROBDD for each constraint.

Abio et al. show that, since all coefficients in all PB constraints are equal to the power of two $2^0 = 1$, the ROBDD that describes the constraint relative to v_i contains only $O(|N(v_i)|^2)$ nodes. Also, since PB constraints are *monotonic* functions, we can translate their respective ROBDDs to CNF by creating only one new variable and two clauses per node. Consider a node of a ROBDD to be represented by the variable y_s. If its selector variable is x_s, its *false* child is represented by the variable f_s, and its *true* child is represented by the variable t_s, then the node can be encoded with the following two clauses: $\neg f_s \rightarrow \neg y_s$, and $(\neg t_s \wedge x_s) \rightarrow \neg y_s$ [13].

The variables representing the root of each ROBDDs are then used to compose the entire formula. Given this formula that codifies a path between vertices in the graph, the resulting MaxSAT instance is built as with the Path-based encodings.

VII. Experimental Results

We implemented all encodings described in this paper, except for the Recursive-Counter-based one, whose source code was made available previously. Our implementations can be downloaded at http://www.inf.ufpr.br/rtoliveira/.

We encoded random instances of the Steiner Tree Problem and some instances from the SteinLib benchmark [10]. They were then solved by MiniMaxSAT [2] on a *AMD Opteron(tm) Processor 6136, 2.4 Ghz, 120 Gb RAM, Linux 3.11.10*.

Tables I and II show the time taken by the solver to solve each instance, in seconds. The columns *DFT, APB, RPB, PrB, RC* and *ROBDD* indicate the DFT-based, Absolute Path-based, Relative Path-based, Parental-based, Recursive-Counter-based and ROBDD-based encodings, respectively. The value *TLE* (*Time Limit Exceeded*) indicates that the instance was not solved within 1800 seconds.

In table I, each instance $V|E|S$ is a random weighted graph with V vertices, E edges and S terminals. In table II, the column $V|E|S$ describes the size of each instance. None of these instances were solved within the given time limit using the DFT-based encoding. Hence, the column DFT was omitted from this table.

By analyzing the results, it is possible to notice that the DFT-based encoding is the least efficient one. It is worth mentioning, however, that its authors suspected this encoding would indeed not be practical [3].

TABLE I. EXPERIMENTAL RESULTS FOR RANDOM INSTANCES

Instance	DFT	APB	RPB	PrB	RC	ROBDD
10|23|4	17.21	0.83	0.01	**0.00**	0.02	0.01
10|23|5	1052.31	7.72	0.27	**0.08**	0.10	14.90
11|28|5	TLE	1.01	0.36	**0.01**	0.18	0.03
12|33|6	TLE	3.03	3.12	0.33	1.09	**0.15**
15|35|8	TLE	1243.84	168.85	**1.02**	4.63	101.80
20|45|13	TLE	TLE	TLE	**4.77**	196.98	TLE
25|54|15	TLE	TLE	TLE	**1.19**	742.90	TLE
30|70|17	TLE	TLE	TLE	**265.48**	TLE	TLE

TABLE II. EXPERIMENTAL RESULTS FOR INSTANCES FROM STEINLIB

Instance	$V\|E\|S$	APB	RPB	PrB	RC	ROBDD
b01	50|63|9	TLE	76.70	2.80	**0.37**	18.33
b02	50|63|13	TLE	1082.63	**3.63**	26.58	554.63
b03	50|63|25	TLE	TLE	**2.57**	13.52	1233.80
b07	75|94|13	TLE	TLE	97.90	**59.66**	653.11
b08	75|94|19	TLE	TLE	**55.63**	TLE	TLE
i080-001	80|120|6	TLE	TLE	TLE	**221.91**	TLE
i080-002	80|120|6	TLE	TLE	TLE	**2.02**	703.87
i080-003	80|120|6	TLE	TLE	TLE	**279.29**	TLE
i080-004	80|120|6	TLE	TLE	TLE	**218.62**	TLE
es20fst03	27|26|20	1.50	3.56	0.02	**0.01**	0.02
es20fst06	29|28|20	1.66	5.47	0.03	**0.01**	0.03
es20fst07	45|59|20	TLE	TLE	**22.72**	1037.00	TLE
es20fst09	36|42|20	TLE	445.77	**0.46**	3.09	157.60
es20fst11	33|36|20	100.39	26.53	0.12	**0.02**	0.31
es30fst07	53|64|30	TLE	TLE	TLE	**57.82**	TLE
es30fst09	43|44|30	TLE	1212.11	0.82	**0.11**	28.38
es30fst10	48|52|30	1199.07	301.70	**0.51**	0.58	105.06
es30fst12	46|48|30	296.67	238.64	0.20	**0.02**	2.66

Also, we can observe that relative encodings are, overall, more efficient than absolute ones. This could be due the fact that relative encodings explicitly codifies how the vertices are related to each other, and not related to a fixed structure. In this sense, relative encodings may contain more relevant information than absolute ones, since paths and trees can be described by the structure these relationships represent.

Surprisingly, the Recursive-Counter-based encoding showed to be, overall, more efficient than the ROBDD-based one. It is believed that encodings of adders and counters have weak propagation properties due to excessive use of XOR gates. Also, the ROBDD-based encoding has stronger proprieties, like being *generalized arc-consistent*: if some variable must be set to false in order to satisfy a formula, then the solver's unit propagation procedure will assign such value [13]. We suspect the decomposition of each cardinality operator to at most three PB constraints may have influenced the efficiency of this encoding.

We can also notice that there is not an overall best encoding among the ones we described. The Parental-based encoding showed to be the most efficient encoding for some instances, while the Recursive-Counter-based one showed to be the most efficient for some others, such as the I080 class of instances from SteinLib. We suspect there are graphs with specific properties that make an encoding more efficient than another.

VIII. CONCLUSION AND FUTURE WORK

In this paper we reviewed two known encodings for trees and suggested four new ones. We described two absolute encodings, two relative encodings and other two counter-based encodings. Our experimental results strongly suggest that relative encodings are more efficient than absolute ones.

As noticed, there was not an overall best encoding among the ones we described. The Recursive-Counter-based encoding showed to be more efficient than the Parental-based one for some instances from SteinLib. As future work, we suggest to characterize these instances to extract their properties that make them easier to solve with a specific encoding. Knowing these characteristics, one can, for instance, create a heuristic to choose the encoding to be used to solve a given instance.

REFERENCES

[1] R. Tavares de Oliveira, F. Silva, B. Ribas, and M. Castilho, "On modeling connectedness in reductions from graph problems to extended satisfiability," in *Advances in Artificial Intelligence IBERAMIA 2012*, ser. Lecture Notes in Computer Science, J. Pavn, N. Duque-Mndez, and R. Fuentes-Fernndez, Eds. Springer Berlin Heidelberg, 2012, vol. 7637, pp. 381–391.

[2] F. Heras, J. Larrosa, and A. Oliveras, "Minimaxsat: An efficient weighted max-sat solver," *Journal of Artificial Intelligence Research*, vol. 31, pp. 1–32, 2008.

[3] H. Kautz, B. Selman, and Y. Jiang, "A general stochastic approach to solving problems with hard and soft constraints," in *The Satisfiability Problem: Theory and Applications*. American Mathematical Society, 1996, pp. 573–586.

[4] F. Mugrauer and A. Balint, "Sat encoded graph isomorphism (gi) benchmark description," in *Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions*, vol. B-2013-1. University of Helsinki, 2013.

[5] M. Samer and H. Veith, "Encoding treewidth into sat," in *Theory and Applications of Satisfiability Testing - SAT 2009*, ser. Lecture Notes in Computer Science, O. Kullmann, Ed. Springer Berlin Heidelberg, 2009, vol. 5584, pp. 45–50.

[6] M. Velev and P. Gao, "Efficient sat techniques for relative encoding of permutations with constraints," in *AI 2009: Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, A. Nicholson and X. Li, Eds. Springer Berlin Heidelberg, 2009, vol. 5866, pp. 517–527.

[7] M. N. Velev and P. Gao, "Efficient sat techniques for absolute encoding of permutation problems: Application to hamiltonian cycles," in *Symposium on Abstraction, Reformulation, and Approximation (SARA)*. AAAI, 2009.

[8] S. Prestwich, "Sat problems with chains of dependent variables," *Discrete Applied Mathematics*, vol. 130, no. 2, pp. 329–350, Aug. 2003.

[9] R. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, ser. The IBM Research Symposia Series, R. Miller, J. Thatcher, and J. Bohlinger, Eds. Springer US, 1972, pp. 85–103.

[10] T. Koch, A. Martin, and S. Voš, "Steinlib: An updated library on steiner tree problems in graphs," Zuse-Institut Berlin (ZIB), Tech. Rep. 00-37, November 2000.

[11] A. V. Gelder, "Another look at graph coloring via propositional satisfiability," *Discrete Applied Mathematics*, vol. 156, no. 2, pp. 230 – 243, 2008, computational Methods for Graph Coloring and its Generalizations.

[12] D. E. Muller and F. P. Preparata, "Bounds to complexities of networks for sorting and for switching," *Journal of the ACM*, vol. 22, no. 2, pp. 195–201, Apr. 1975.

[13] I. Abío, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and V. Mayer-Eichberger, "A new look at bdds for pseudo-boolean constraints," *Journal of Artificial Intelligence Research*, vol. 45, no. 1, pp. 443–480, Sep. 2012.