On a Relative MaxSAT Encoding for the Steiner Tree Problem in Graphs

Ricardo Tavares de Oliveira and Fabiano Silva

Departamento de Informática, Universidade Federal do Paraná
POBox 19081, 81531-980, Curitiba, Brazil
{rtoliveira,fabiano}@inf.ufpr.br
http://www.inf.ufpr.br

Abstract. In [1] it was presented some MaxSAT encodings for trees in graphs which can be used to solve the Steiner Tree Problem. In this paper we focus exclusively on the relative encoding which was called *Parental-based*. We review this encoding and improve it by applying two techniques. One of them is a known improvement to encode transitivity, previously used for other relative encodings. The other one consists on deducing unit clauses from the dominance relation of the given graph. Finally, we use the improved encodings to solve relevant instances, and present experimental results.

Keywords: Boolean satisfiability, SAT encodings, MaxSAT encodings, relative encoding, Steiner Tree

1 Introduction

The efficiency of SAT and MaxSAT solvers has grown in the last decades. This motivates solving relevant problems by encoding them to (Max)SAT and providing the resulting formulae as input to state-of-art (Max)SAT solvers.

Many encodings of relevant problems are known [1–4]. In this paper, we focus on an encoding for the *Steiner Tree Problem*. This problem is known to be NP-Hard [5], and has applications, for instance, in Computational Geometry and Circuit Design [6].

Previously, it was presented different MaxSAT encodings for the Steiner Tree Problem [1], which were classified as *absolute*, *relative* or *counting-based*. In this paper, we review and improve the relative *Parental-based* encoding, the most efficient encoding among the relative ones presented previously.

We apply two improvements on this encoding. One of the procedures we use to improve the encoding was previously used by Bryant & Velev [3] and Velev & Gao [4] to encode transitivity for other problems. The other one consists on using the dependence relation of the given graph to deduce unit clauses.

This paper is organized as follows: Section 2 provides preliminary definitions and notations. Section 3 provides a brief background on SAT and MaxSAT encodings for problems in graphs. Section 4 reviews the *Parental-based* encoding. Section 5 presents the first improvement on the encoding, while section 6 presents the second one. Finally, section 7 shows experimental results, while section 8 concludes and present future works.

2 Preliminaries

In this section we present some preliminaries notions and definitions. First, the *Steiner Tree Problem* consists in, given a weighted graph $G = (V, E, w : E \rightarrow \mathbb{N}^+)$ and a set of vertices $S \subseteq V$ (the *terminal* vertices), find a connected subgraph of G containing all terminal vertices whose sum of the weight of its edges is minimized. Such subgraph is clearly a tree.

A *(Boolean) variable* can assume either 0 (false) or 1 (true). A *literal* is a variable x_i or its negation $\neg x_i$. A *clause* is a disjunction ($\vee$) of literals. The expression $A \rightarrow B$ denotes $(\neg A) \vee B$, which always results in a clause in this text. An *unit* clause is a clause containing exactly one literal, and an *empty* clause is a clause containing no literals, which is always evaluated to 0 (false). A *Conjective Normal Form (CNF)* formula is a conjunction ($\wedge$) of clauses.

An *assignment* is a set of literals where a variable and its negation do not occur simultaneously in it. Given a CNF formula, an assignment A is *total* if $x_i \in A$ or $\neg x_i \in A$ for all variable x_i occurring in the formula, and *partial* otherwise. An assignment A is an *extension* of a partial assignment A' (or, equivalently, A' can be *extended to* A) if $A' \subset A$. An assignment A *satisfies* a clause C if there is a literal in both the assignment A and the clause C. An assignment satisfies a CNF formula if it satisfies all its clauses. A total assignment that satisfies a CNF formula is a *model* of it. The *Boolean Satisfability Problem* (SAT) consists in, given a CNF formula, decide whether it has a model.

Unit Propagation is a procedure used by most state-of-the-art SAT solvers to simplify a CNF formula [7, 8]. If a given CNF formula contains an unit clause (x_i) (resp. $(\neg x_i)$), then the procedure *propagates* the literal x_i (resp. $\neg x_i$) in the formula, i.e., it replaces each occurrence of the variable x_i by 1 (true) (resp. 0 (false)). The procedure is repeated until the formula does not contain any unit clause or contains an empty clause.

Let A be a partial assignment (possibly empty) which can be extended to a model of a given CNF formula, and consider that all literals in A are propagated in such formula. The resulting formula is *Generalized Arc-Consistent (GAC)* if there is not a variable x_i such that: (i) $A \cup \{\neg x_i\}$ can be extended to a model of the formula; (ii) $A \cup \{x_i\}$ can *not* be extended to a model of the formula; (iii) $\neg x_i \notin A$. Informally, the formula is GAC if all variables that must be set to 0 (false) to make the formula satisfiable are present in the current partial assignment, and thus are not present in the resulting formula at all. Also, the given formula is *maintained GAC by Unit Propagation* if the literal $\neg x_i$ is propagated by such procedure, for all variable x_i such that $A \cup \{\neg x_i\}$ can be extended to a model of the formula, but $A \cup \{x_i\}$ can not, for all such partial assignments A during the search.

The *Partial Weighted Maximum Boolean Satisfability Problem* ((PW)MaxSAT) consists in, given a CNF formula F_h (the *hard* formula), a set of clauses F_s (the *soft* clauses), and a function $W : F_s \rightarrow \mathbb{N}^+$ (the *weight* or *cost* of each *soft* clause), find a model of the *hard* formula whose sum of the weight of the satisfied *soft* clauses is maximized. Unit Propagation can also be used by MaxSAT solvers to simplify the *hard* formula [9].

Finally, given a graph $G = (V, E)$, we denote by $N(v_i) = \{v_j \in V | \{v_i, v_j\} \in E\}$ the set of neighbors of a given vertex v_i.

3 Background

In this section we give a brief background about SAT and MaxSAT encodings for problems in graphs.

There is more than one way to reduce a given problem to SAT or MaxSAT. A possible way to do so is by describing the problem as a Constraint Satisfiability Problem (CSP) and then encode its constraints into CNF formulae. These encodings are referred as *absolute* encodings. This notation was used by Prestwich [2] to describe an encoding for the Hamiltonian Cycle Problem. In his encodings, a permutation of the vertices in the given graph, which described the path, is encoded. In the absolute encoding, there is a boolean variable for each vertex in the given graph and each position of the permutation. Many distinct absolute encodings can be used, such as the *direct*, *muldirect*, and *log* encodings. For a review of these encodings, the reader may referrer to Velev [10].

For some problems, one can describe the problem as a binary relation instead of a CSP. The encodings that describe a binary relation are referred as *relative* encodings. This notation was also used by Prestwich [2] to describe another encoding for the Hamiltonian Cycle Problem. In this encoding, each boolean variable states the relative positions, in the described permutation, between vertices in the given graph.

In relative encodings, it is usually necessary to describe a *transitive* relation. The transitivity property can naturally be encoded by a cubic number of clauses in the form $(r_{a,b} \land r_{b,c}) \to r_{a,c}$, where $r_{a,b}$ states the relation between elements a and b. Bryant & Velev [3] suggested an improvement for this property in particular. Velev & Gao [4] then improved Prestwich's encoding. This improvement is shown in section 5 applied to the *Parental-based* encoding.

Previously [1], it was presented, among others, two relative encodings for the Steiner Tree Problem to (PW)MaxSAT. In this work, we focus on the *Parental-based* encoding. The encoding describes the partial (binary) relation induced by a tree in the given graph. This encoding is reviewed in the next section.

4 The ***Parental-based*** encoding

In this section we review the *Parental-based* encoding. Given a graph $G = (V, E)$ and a set of its vertices $S \subseteq V$, the *Parental-based* encoding creates a *hard* formula $F_{PrB}(G, S)$ which is satisfiable if and only if all verticies in S are in the same connected component of G [1]. Also, each model of $F_{PrB}(G, S)$ describes a tree in G containing all vertices in S. To solve the Steiner Tree Problem, *soft* clauses are then created to minimize such tree [1].

The encoded tree is rooted in an arbitrary terminal vertex $v_r \in S$. The root of the tree is chosen among the terminal vertices during a pre-processing step. We suggest seven heuristics to select such vertex: (1) Select the first terminal vertex in the input file; (2) Select a terminal vertex with maximum degree. Break ties by selecting the first such vertex in the input file; (3) Select a terminal vertex with maximum degree. Break ties by randomly selecting one such vertex; (4) Select a terminal vertex with minimum degree. Break ties by selecting the first such vertex in the input file; (5) Select a terminal vertex with minimum degree. Break ties by randomly selecting one such vertex; (6) Select a terminal vertex whose average distance to all others terminal vertices is minimum. Break ties by randomly selecting one such vertex; (7) Select a terminal vertex whose average distance to all others terminal vertices is maximum. Break ties by randomly selecting one such vertex. One can use the Floyd-Warshall algorithm [11] to compute the distances used by heuristics 6 and 7.

The formula $F_{PrB}(G, S)$ is built as follows [1]: for each edge $\{v_i, v_j\} \in E$, two boolean variables $p_{i,j}$ and $p_{j,i}$ are created. The variable $p_{i,j}$ (resp. $p_{j,i}$) states that v_j (resp. v_i) is the *parent* of v_i (resp. v_j) in the described tree.

Also, two other boolean variables $a_{i,j}$ and $a_{j,i}$ are created for each pair of distinct vertices $v_i, v_j \in V$. The variable $a_{i,j}$ (resp. $a_{j,i}$) states that v_j (resp. v_i) is an *ancestor* of v_i (resp. v_j) in the described tree. It is worth noticing that variables $p_{i,j}$ encode a *binary relation* P over the vertices of the given graph, while variables $a_{i,j}$ encode its *transitive closure* P^+.

Finally, another boolean variable $y_{i,j}$ is created for each edge $\{v_i, v_j\} \in E$. The variable $y_{i,j}$ states that the edge $\{v_i, v_j\}$ is present in the described tree.

The *hard* formula $F_{PrB}(G, S)$ contains eight types of clauses:

(*terminal-presence*) $(\bigvee_{v_j \in N(v_s)} p_{s,j})$ for each $v_s \in S, v_s \neq v_r$. These clauses ensure that all terminal vertices, except for the root, must have a parent in the tree, and thus are present in the described subgraph;

(*at-most-one-parent*) $(p_{i,j} \rightarrow \neg p_{i,k})$ for each $v_i \in V, v_i \neq v_r$ and for each pair of distinct vertices $v_j, v_k \in N(v_i)$. These clauses ensure that no vertex has more than one parent in the tree;

(*connectedness*) $(p_{j,i} \rightarrow \bigvee_{v_k \in N(v_i)} p_{i,k})$, for each $v_i \in V, v_i \neq v_r$ and for each $v_j \in N(v_i), v_j \neq v_r$. These clauses ensure that, if a given vertex has a parent in the tree, then its parent also has a parent in such tree, except for the root;

(*subset*) $(p_{i,j} \rightarrow a_{i,j})$ for each $v_i \in V, v_i \neq v_r$ and for each $v_j \in N(v_i)$. These clauses state that if a vertex is the parent of another vertex in the tree, then it is also one of its ancestors. This encodes $P \subseteq P^+$;

(*transitivity*) $((a_{i,j} \wedge a_{j,k}) \rightarrow a_{i,k})$ for each triple of distinct vertices $v_i, v_j, v_k \in V$. These clauses encode the transitivity of the ancestor relation P^+;

(*asymmetry*) $(a_{i,j} \rightarrow \neg a_{j,i})$ for each pair of distinct vertices $v_i, v_j \in V$. These clauses state that the ancestor relation P^+, and thus also the parental relation P, is asymmetric;

(*root-path*) $(a_{s,r})$ for all $v_s \in S, v_s \neq v_r$. These unit clauses state that the root of the tree is an ancestor of all other terminal vertices;

(*edge-vertex-relation*) $(y_{i,j} \leftrightarrow (p_{i,j} \vee p_{j,i})) = (y_{i,j} \rightarrow (p_{i,j} \vee p_{j,i}))$, $(p_{i,j} \rightarrow y_{i,j})$ and $(p_{j,i} \rightarrow y_{i,j})$, for each edge $\{v_i, v_j\} \in E$. These clauses state that an edge is present in the tree iff one of its vertices is the parent of its other one.

Finally, an unit *soft* clause $(\neg y_{i,j})$ with weight $w(\{v_i, v_j\})$ is created for each edge $\{v_i, v_j\} \in E$, stating that the sum of the weights of the edges in the tree must be minimized.

This encoding creates $O(|V|^2)$ boolean variables, $O(|V|^3)$ *hard* clauses and $|E|$ *soft* clauses. The total number of literals in both the *hard* and *soft* clauses is also in $O(|V|^3)$. Asymptotically, this encoding creates the smaller number of variables, clauses and literals among the Path-based encodings presented in [1]. Also, it is worth noticing that the largest portion of the instance is due the encoding of the *transitivity* property, which requires a cubic number in $|V|$ of *hard* clauses. In the next section we present an improvement on this part of the formula in particular.

It is also worth observing that a model A of the *hard* formula may describe a subgraph $G'(A)$ containing more vertices and edges than the ones presented in the described tree. These elements are removed from the solution exclusively via MaxSAT optimization.

5 An improvement on the transitivity relation

As previously stated, this encoding creates a *hard* formula with $O(|V|^2)$ variables, $O(|V|^3)$ clauses and $O(|V|^3)$ literals in total, and $|E|$ unit *soft* clauses.

The number of variables and clauses in the *hard* formula can be reduced using a method described by Bryant & Velev [3] and applied to the Hamiltonian Cycle Problem by Velev & Gao [4]. Their method is based on the fact that the transitive relation between some pairs of vertices is not directly relevant to the solution, and thus some variables may be omitted from the formula.

Let us define the *relational graph* as the graph $G_R = (V, E \cup \{\{v_r, v_s\} | v_s \in S, v_s \neq v_r\})$, i.e., G_R is the graph G with additional edges connecting the root v_r and all other terminal vertices in S (if not present already). Instead of defining variables $a_{i,j}$ and $a_{j,i}$ for each pair of distinct vertices $v_i, v_j \in V$, we define these variables only for each pair of vertices where $\{v_i, v_j\} \in E(G_R)$. In practice, we remove from the encoding the variables that originally occur in the *transitivity* and *asymmetry* clauses *only*. These variables are not directly relevant to the encoding and their values could be inferred from other variables in the solution.

Transitivity can then be encoded by enumerating all chord-free cycles in G_R, as suggested by Bryant & Velev [3]. For each chord-free cycle with k vertices, k clauses are added to the formula. Each clause states the relation between the vertices in one edge and the vertices in all the other edges in the cycle.

As also suggested by Bryant & Velev [3], the transitivity property can be encoded efficiently if every chord-free cycle in G_R is a triangle with 3 vertices, i.e., if G_R is *chordal*. If G_R is not chordal, it is possible to add a set of edges (called a *fill*) to the graph to make it chordal. It is NP-Hard to obtain a fill with the smallest possible number of edges [12]. However, some heuristics can be used to obtain a “good” set of edges.

Velev & Gao [4] presented the following procedure to obtain such a set: Let G^+ be a graph initially equal to G_R, and let F be an initially empty set. Select a vertex $v_i \in G^+$ and, for each pair of distinct vertices v_j, v_k such that

$\{v_i, v_j\} \in E(G^+)$ and $\{v_i, v_k\} \in E(G^+)$, add the edge $\{v_j, v_k\}$ to both the set F and the graph G^+, if not present already. Then, remove the vertex v_i and all its incident edges from G^+. Repeat the procedure until G^+ is empty. At the end, include all edges in F to G_R to make it chordal [4].

Twelve heuristics to choose a vertex at each step are known [4]: (1) Select a vertex in G^+ with minimum degree. Break ties by selecting the vertex whose sum of the degrees of its neighbors is minimum; (2) Select a vertex in G^+ with minimum degree. Break ties by selecting the vertex whose sum of the degrees of its neighbors is maximum; (3) Select a vertex in G^+ with minimum degree. Break ties by selecting the vertex whose number of edges to be added at that step of the procedure, if that vertex is selected, is minimum; (4) Select a vertex in G^+ with minimum degree. Break ties by selecting the vertex whose number of edges to be added at that step of the procedure, if that vertex is selected, is maximum; (5) Select a vertex in G^+ with minimum degree. Break ties by selecting one such vertex whose degree in G_R is minimum; (6) Select a vertex in G^+ with minimum degree. Break ties by selecting one such vertex whose degree in G_R is maximum; (7) Select a vertex in G^+ with minimum degree. Break ties by selecting the vertex that, if selected, minimizes the number of triangles, in the graph given by the union of G_R with the edges currently in F, containing the selected vertex and some edge in F; (8) Select a vertex in G^+ with minimum degree. Break ties by selecting the vertex that, if selected, maximizes the number of triangles, in the graph given by the union of G_R with the edges currently in F, containing the selected vertex and some edge in F; (9) Select a vertex in G^+ that, if selected, minimizes the number of edges to be added to G^+ at that step of the procedure. Break ties by selecting the first such vertex in the input file; (10) Select a vertex in G^+ that, if selected, minimizes the number of edges to be added to G^+ at that step of the procedure. Break ties by randomly selecting one such vertex; (11) Select a vertex to G^+ that, if selected, minimizes the number of triangles, in the graph given by the union of G_R with the edges currently in F, containing the selected vertex and some edge in F. Break ties by selecting the first such vertex in the input file; (12) Select a vertex to G^+ that, if selected, minimizes the number of triangles, in the graph given by the union of G_R with the edges currently in F, containing the selected vertex and some edge in F. Break ties by randomly selecting one such vertex. For all heuristics, if not stated otherwise, if there are still ties, break it by selecting the first vertex in the input that matches the given criteria.

We define variables $a_{i,j}$ and $a_{j,i}$ only for pairs of vertices $v_i, v_j \in V$ that are adjacent in G_R. Also, a *transitivity* clause $((a_{i,j} \wedge a_{j,k}) \rightarrow a_{i,k})$ is added only when all variables $a_{i,j}$, $a_{j,k}$ and $a_{i,k}$ are defined. Since G_R is chordal, a *transitivity* clause is created for each triangle in this graph. The number of triangles in G_R may be way smaller than the number of *transitivity* clauses created in the original encoding, which is near to $|V|^3$.

Finally, a *asymmetry* clause $(a_{i,j} \rightarrow \neg a_{j,i})$ is also added only when the variables $a_{i,j}$ and $a_{j,i}$ are defined. The number of *asymmetry* clauses created is equal to the number of edges in G_R, which is in $O(|E| + |S| + |F|)$, the sum of the number of edges in the original graph, the number of edges connecting the root to all terminal vertices, and the number of edges in F.

As stated by Bryant & Velev [3], this improvement can be applied without invalidating the correctness of the transitivity encoding. Indeed, the solutions found during our experiments are all optimal according to benchmark descriptions and previous experiments.

6 On deducing unit clauses from the dominance relation

In this section we suggest another improvement on the *Parental-based* encoding.

It is possible to anticipate a truth value for some variables before starting the MaxSAT solver by analyzing the input graph. Informally, this improvement consists on *deducing unit clauses* based on the original problem and the meaning of the variables in the encoding.

Let $G' = (V, E')$ be the directed graph obtained by replacing each edge in the original graph G by two directed arcs, i.e., $E' = \{(v_i, v_j), (v_j, v_i) | \{v_i, v_j\} \in E\}$. Also, let v_r be the terminal vertex selected to be root of the encoded tree, as defined in section 4.

The *dominator tree* D of G' w.r.t. v_r is a tree rooted at v_r such that, if a vertex v_i is an ancestor of a vertex v_j in D, then every path from v_r to v_j in G contains the vertex v_i [13]. Hence, if v_i is an ancestor of v_j in D, then it *not* possible to obtain a tree in G such that v_j occurs *before* v_i in a path starting in v_r. Thus, v_j cannot be an ancestor of v_i in the described tree, so we can deduce that the variable $a_{i,j}$, if defined, must be set to 0 (false). In this case, we create the unit clause $(\neg a_{i,j})$ and add it to the *hard* formula.

If the vertex v_j is present in the tree described by a model of the *hard* formula, then the literal $a_{j,i}$ is certainly present in such model. However, there is a model containing $a_{j,i}$ even if the vertex v_j is not present in the encoded tree. Notice that the *subset* clause $(p_{j,i} \rightarrow a_{j,i})$ is satisfied in this case even if $p_{j,i}$ is set to 0 (false). Hence, it is also possible to add the unit clause $(a_{j,i})$ to the formula.

We use the Lengauer-Tarjan algorithm [13] to build the dominator tree. Then, for each pair of vertex v_i, v_j such that v_i is an ancestor of v_j in the dominator tree and the variables $a_{i,j}$ and $a_{j,i}$ are defined, we add the unit clauses $(\neg a_{i,j})$ and $(a_{j,i})$ to the *hard* formula. These literals will be propagated by the Unit Propagation procedure as the solver starts.

It is worth mentioning that the addition of the unit clause $(\neg a_{i,j})$ makes Unit Propagation propagate the literal $\neg p_{i,j}$ (if defined), due to the *subset* clause $(p_{i,j} \rightarrow a_{i,j})$. In fact, we conjecture that, with this improvement, Unit Propagation makes the *hard* formula *Generalized Arc-Consistent* (GAC), i.e., if there is any variable that must be set to 0 (false) in order to make the formula satisfied, then Unit Propagation will propagate such assignment. However, this fact may be valid only for the formula given as input to the solver – GAC is not *maintened* by Unit Propagation during the search. This maintenance is discussed as a future work in section 8.

It is also worth noticing that, although we applied this technique on the *Parental-based* encoding in particular, this improvement may actually be applied on any relative encoding that encodes the transitivity property.

7 Experimental Results

In this section we present some experimental results obtained by solving relevant instances of the Steiner Tree Problem using the presented encoding.

We used the encoding to reduce some random instances of the Steiner Tree Problem and instances from the SteinLib benchmark [6]. The random instances used in our experiments, namely *20_45_13, 25_54_15, 30_70_17* and *35_98_19*, as well as all source codes of all tools used in this paper, can be downloaded at *http://www.inf.ufpr.br/rtoliveira/*.

To efficiently encode a given instance of the problem, it is needed to determine (i) the root v_r of the encoded tree and (ii) the vertex to be selected at each step of the procedure used to make G_R chordal, presented in section 5. We consider seven heuristics for the selection of the root and twelve for the selection of such vertex, as presented in sections 4 and 5.

First, we encoded the instances with the improvement on the transitivity property using all combinations of both heuristics. We then compared the size of the resulting formulae against the size of the formulae obtained by the *Parental-based* encoding *as-is*, i.e., as presented in [1].

Table 1 shows the results. The column *PrB* stands for the *Parental-based* encoding *as-is*, while *IPrB* stands for the improved version of the encoding, with the improvement on the transitivity property. Columns *Vars* indicate the number of variables in the formulae, while columns *Claus* indicate the number of clauses in them. The columns *h(i)* and *h(ii)* indicate which heuristic to (i) select the root and (ii) select the vertex at each step of the procedure resulted in the smaller number of variables and clauses in the formulae.

Table 1. Size of the formulae generated by the encodings

Encoding	**PrB**		**IPrB**			
Instance	Vars	Claus	Vars	Claus	h(i)	h(ii)
20_45_13	511	7750	**279**	**1543**	3	1
25_54_15	758	14920	**382**	**2371**	7	2
30_70_17	1076	25937	**507**	**3670**	2	2
35_98_19	1480	41663	**743**	**7035**	7	2
i080-001	6673	497740	**817**	**4246**	1	2
i080-002	6676	497772	**860**	**4766**	3	2
i080-003	6672	497736	**840**	**4368**	1	2
i080-004	6674	497768	**858**	**4870**	1	10
i080-005	6677	497732	**904**	**5487**	2	2

Encoding	**PrB**		**IPrB**			
Instance	Vars	Claus	Vars	Claus	h(i)	h(ii)
es30fst01	6505	479011	**879**	**3897**	3	3
es30fst02	5259	346623	**712**	**2828**	7	3
es30fst03	7162	556189	**904**	**3994**	3	10
es30fst04	6664	497572	**872**	**3767**	7	3
es30fst05	3517	187649	**531**	**1957**	2	3
es30fst07	2946	142714	**464**	**1638**	2	2
es30fst08	4968	317846	**676**	**2584**	7	10
es30fst09	1937	75471	**310**	**936**	2	1
es30fst10	2411	105526	**355**	**1081**	3	2
es30fst11	6496	478955	**858**	**3739**	3	10
es30fst12	2213	92691	**321**	**928**	2	1
es30fst13	4410	265192	**630**	**2471**	2	10
es30fst14	2929	142638	**389**	**1177**	2	2

As expected, this improvement on the encoding reduced the size of the resulting formulae. For the instance *i080-001*, the number of variables was reduced by aprox. 8 times, while the number of clauses was reduced by aprox. 117 times.

It is also worth noticing that the selections 2, 3 and 7 showed to be the best heuristics overall to choose the root of the tree.

We then encoded all instances: with the improvement on the transitivity property only; with the improvement on deducing unit clauses only; and with both improvements. In the cases where the first improvement applies, we considered all 12 heuristics to choose the vertex to be selected at each step of the procedure, and the heuristics 2, 3 and 7 to select the root of the tree. In the case where only the second improvement applies, we used the heuristic 1 to select the root of the encoded tree, as implemented by [1]. The resulting formulae were given as input to MaxSAT solvers MiniMaxSAT (`minimaxsat1.0`) [9] and EvaSolver (`eva500a__`) [14]. We then ran the solvers on an *AMD Opteron(tm) Processor 6136, 2.4 Ghz, 120 Gb RAM, Linux 3.16.7.*

Table 2 shows the best obtained results. *PrB* stands for the *Parental-based* encoding *as-is*; *IPrB* stands for the improved version of the encoding, with the improvement on the transitivity property only; *UPrB* stands for the improved version with deduced unit clauses only; *UIPrB* stands for the encoding improved by both improvements. In the cases where the first improvement applies, each instance was encoded 10 times for each combination of heuristics. Column *CPU* indicates the average CPU time took by the solver in seconds, while column σ indicates its standard deviation. TLE (*Time Limit Exceeded*) indicates that the given formula was not solved within 1800 seconds. The symbol $*$ indicates that there was not an unique best combination of heuristics for that instance.

Let us first analyze the results obtained by the improvement on the transitivity property only (*IPrB*). As it can be observed, except for isolated cases, this improvement on the encoding impacted significantly on the run time taken by solvers to solve the obtained formulae. Indeed, some instances previously unsolved by MiniMaxSAT with the *Parental-based* encoding, such as the ones in the class *I080*, can be solved with the improved encoding by the same solver.

It is also worth observing that the combination of heuristics that generates the smaller formulae is not necessarily the combination that generates the easier formulae. It is also interesting noticing that there is not an overall best MaxSAT solver to solve these instances. EvaSolver performed better than MiniMaxSAT for some instances, mainly for the class *ES30FST*, while MiniMaxSAT performed better than EvaSolver in others. This may indicate there is a relation between the instances' characteristics and the internal algorithms and heuristics used by the solvers.

Let us then analyze the results obtained by the improvement by deducing unit clauses only (*UPrB*). We expected this improvement to make the formulae easier to solve. Surprisingly, although this improvement did make the solvers solve specific instances faster, it did *not* improved their overall run time. In fact, this improvement made some instances actually *harder* to be solved.

To help us to investigate this fact, we combined the second improvement with the $12 \times 3 = 36$ combinations of heuristics used for the experiments for the first improvement, and analyzed the cases where the second improvement made the resulting formulae easier or harder to solve.

Table 2. Best results for both solvers with the first improvement, with the second one and with both

Encoding	**PrB**	**IPrB**				**UPrB**	**UIPrB**			
	CPU	CPU	σ	h(i)	h(ii)	CPU	CPU	σ	h(i)	h(ii)
Solver	MiniMaxSAT [9]									
20_45_13	4.77	**0.64**	0.00	7	2	6.97	0.78	0.00	7	3
25_54_15	1.19	0.68	0.00	3	5	6.06	**0.56**	0.00	2	2
30_70_17	265.48	18.28	0.00	2	1	273.88	**12.92**	0.00	7	1
35_98_19	TLE	TLE	-	-	-	TLE	TLE	-	-	-
es30fst01	TLE	TLE	-	-	-	TLE	TLE	-	-	-
es30fst02	TLE	TLE	-	-	-	TLE	TLE	-	-	-
es30fst03	TLE	TLE	-	-	-	TLE	TLE	-	-	-
es30fst04	TLE	TLE	-	-	-	TLE	TLE	-	-	-
es30fst05	TLE	35.71	0.00	7	2	TLE	**23.01**	0.00	7	2
es30fst07	TLE	1.20	0.00	2	3	TLE	**0.95**	0.00	2	4,5
es30fst08	TLE	TLE	-	-	-	TLE	TLE	-	-	-
es30fst09	0.82	**0.01**	0.00	*	*	0.33	**0.01**	0.00	*	*
es30fst10	0.52	**0.01**	0.00	*	*	0.32	**0.01**	0.00	*	*
es30fst11	TLE	TLE	-	-	-	TLE	TLE	-	-	-
es30fst12	0.20	**0.00**	0.00	*	*	0.33	**0.00**	0.00	*	*
es30fst13	TLE	502.12	1.03	2	2	TLE	**17.77**	0.00	2	6
es30fst14	0.75	**0.00**	0.00	*	*	0.47	**0.00**	0.00	*	*
i080-001	TLE	765.94	1.5	2	1	1292.42	**272.99**	0.45	2	6
i080-002	TLE	281.10	0.55	2	9	920.36	**250.42**	1.33	2	6
i080-003	TLE	**90.97**	0.17	2	5	TLE	205.42	0.46	2	7
i080-004	TLE	197.70	0.46	2	2	TLE	**101.05**	0.20	2	6
i080-005	TLE	1146.93	5.08	2	9	TLE	**304.91**	0.38	3	2
Solver	EvaSolver [14]									
20_45_13	87.74	**2.55**	0.00	7	9	150.01	5.77	1.20	7	10
25_54_15	13.81	**0.22**	0.00	*	*	1.27	0.24	0.00	7	6
30_70_17	TLE	TLE	-	-	-	TLE	TLE	-	-	-
35_98_19	1128.26	**135.55**	3.63	7	4	955.81	236.30	17.34	7	4
es30fst01	TLE	**1698.69**	31.18	2	5	TLE	TLE	-	-	-
es30fst02	172.47	**10.33**	0.00	7	5	160.05	13.05	0.33	7	5
es30fst03	1392.26	23.06	0.14	7	2	896.13	**20.34**	0.20	7	4
es30fst04	TLE	**662.33**	6.30	2	5	TLE	1275.68	44.45	2	6
es30fst05	TLE	TLE	-	-	-	TLE	TLE	-	-	-
es30fst07	9.22	**0.28**	0.00	*	*	10.31	0.30	0.00	*	*
es30fst08	93.14	**8.15**	0.00	3	4	94.66	8.89	0.22	2	4
es30fst09	3.91	**0.24**	0.00	7	10	8.99	0.33	0.00	7	9
es30fst10	2.54	**0.08**	0.00	3	2	3.45	**0.08**	0.00	3	7
es30fst11	TLE	**460.37**	8.57	7	4	TLE	523.47	12.52	7	4
es30fst12	0.70	**0.01**	0.00	*	*	0.80	**0.01**	0.00	*	*
es30fst13	46.22	**2.18**	0.00	*	*	61.22	2.39	0.00	2	4
es30fst14	1.13	**0.01**	0.00	*	*	0.99	**0.01**	0.00	*	*
i080-001	**245.63**	313.09	54.94	7	12	377.24	294.36	9.99	7	1
i080-002	1100.40	**242.45**	4.24	7	9	1169.88	259.70	9.27	7	9
i080-003	119.52	**115.31**	2.94	2	9	143.96	138.22	8.93	3	2
i080-004	**724.11**	954.66	16.33	7	6	753.81	887.64	53.00	2	6
i080-005	688.35	**531.31**	9.92	7	7	887.96	833.94	46.88	7	3

Table 3 shows the results. Column *UIPrB* indicates the number of combinations of heuristics for which the formula obtained by using *both* improvements were solved faster, while column *IPrB* indicates the number of combinations of heuristics for which the formula obtained by using the first improvement only were solved faster. It is worth mentioning that the sum of both values may not add to 36 due to formulae that were not solved within the time limit.

By analyzing table 3, we can notice that, overall, the formulae obtained by using both improvements are better solved by MiniMaxSAT, while the formula obtained by not using the second improvement are better solved by EvaSolver. This

Table 3. Number of combinations that performed better for each instance

Solver	MiniMaxSAT [9]		EvaSolver [14]	
Encoding	**UIPrB**	**IPrB**	**UIPrB**	**IPrB**
20_45_13	**29**	7	6	**30**
25_54_15	**22**	14	11	**20**
30_70_17	**22**	6	0	0
35_98_19	0	0	16	**20**
i080-001	4	4	10	**26**
i080-002	**10**	2	8	**20**
i080-003	**16**	15	6	**30**
i080-004	**21**	6	**9**	3
i080-005	**4**	3	4	**17**

Solver	MiniMaxSAT [9]		EvaSolver [14]	
Encoding	**UIPrB**	**IPrB**	**UIPrB**	**IPrB**
es30fst01	0	0	0	**1**
es30fst02	0	0	16	**20**
es30fst03	0	0	**21**	13
es30fst04	0	0	3	**8**
es30fst05	**30**	0	0	0
es30fst07	**22**	11	3	**32**
es30fst08	0	0	2	**34**
es30fst09	**22**	1	2	**31**
es30fst10	**13**	3	7	**29**
es30fst11	0	0	0	**11**
es30fst12	0	**4**	1	**4**
es30fst13	1	**2**	7	**29**
es30fst14	6	**12**	1	**6**

seems particularity true for the random instances and the class *ES30FST*, where the differences between the respective number of easier instances are larger.

We suspect that the performance obtained with and without the improvement may be related to the base algorithm and to the internal heuristics used by the solvers. MiniMaxSAT is based on a *Branch and Bound* DPLL-like algorithm [9], while EvaSolver is based on successive eliminations of *unsatisifable cores* [14]. Since the base algorithm used by each solver is different, it may be expected that the deduced unit clauses may impact them differently.

Also, the deduced unit clauses may interfere in the internal heuristics used by the solvers. After unit propagation, the resulting formulae may be such that the solver decides to use "worse" heuristics and hence explore the search space poorly, which may not be the case if the formula remains unchanged, without the deduced unit clauses. Further investigation on the solver' internal algorithms and instances' characteristics is needed to confirm this conjecture.

As stated in section 6, we conjecture that the second improvement makes the (initial) formula GAC, but unit propagation does not maintains it during the search. We also conjecture that, if the formula is *maintained* GAC by unit propagation during the search, then the solvers will perform better with the second improvement for *all* instances. We suggest studying such maintenance as a future work, as discussed in section 8.

Finally, let us briefly analyze the results obtained by both improvements (*UIPrB*). Again, it is possible to notice that the second improvement did not make all instances easier as expected, as previous discussed. However, for some particular instances, such as *30_70_17*, *es30fst03*, *es30fst05* and *i080-004*, the combination of both improvements did make the instance easier to be solved.

8 Conclusion and Future Work

In this paper we review the *Parental-based* relative encoding which is used to solve the Steiner Tree Problem in graphs and improve it by using a method described and used previously by Bryant & Velev [3] and Velev & Gao [4], and another one that explore the dominance relation in the given graph.

As shown in section 7, the first method reduced the size of the resulting formulae and the run time taken by MaxSAT solvers to solve them, as expected.

The second method made some instances easier to solve, but did not improved the run times overall.

As mentioned in section 6, we conjecture that the second improvement makes the *hard* formula GAC, but unit propagation does not maintains this property during the search. As mentioned in section 7, we also conjecture that maintaining GAC during search may make the second method always improve the solvers.

Since it is polynomial to decide whether all terminal vertices are in the same connected component, we suspect that it may be possible to build the *hard* formula in such a way that GAC is maintained by unit propagation. As a future work, we suggest studying some encoding for which the *hard* formula is maintained GAC by unit propagation, or prove that such encoding does not exist.

References

1. Oliveira, R.T.d., Silva, F.: Sat and maxsat encodings for trees applied to the steiner tree problem. In: Intelligent Systems (BRACIS), 2014 Brazilian Conference on. (Oct 2014) 192–197
2. Prestwich, S.: Sat problems with chains of dependent variables. Discrete Applied Mathematics **130**(2) (August 2003) 329–350
3. Bryant, R.E., Velev, M.N.: Boolean satisfiability with transitivity constraints. ACM Trans. Comput. Logic **3**(4) (2002) 604–627
4. Velev, M.N., Gao, P.: Design of parallel portfolios for sat-based solving of hamiltonian cycle problems. In: International Symposium on Artificial Intelligence and Mathematics (ISAIM 2010), Fort Lauderdale, Florida, USA, January 6-8, 2010. (2010)
5. Karp, R.: Reducibility among combinatorial problems. In Miller, R., Thatcher, J., Bohlinger, J., eds.: Complexity of Computer Computations. The IBM Research Symposia Series. Springer US (1972) 85–103
6. Koch, T., Martin, A., Voß, S.: Steinlib: An updated library on steiner tree problems in graphs. Technical Report 00-37, Zuse-Institut Berlin (ZIB) (November 2000)
7. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications of the ACM **5** (July 1962) 394–397
8. Biere, A.: Lingeling, plingeling and treengeling entering the sat competition 2013. In: Proceedings of SAT Competition 2013. Volume B-2013-1., University of Helsinki (2013)
9. Heras, F., Larrosa, J., Oliveras, A.: Minimaxsat: An efficient weighted max-sat solver. Journal of Artificial Intelligence Research **31** (2008) 1–32
10. Velev, M.N.: Exploiting hierarchy and structure to efficiently solve graph coloring as sat. In: Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on. (Nov 2007) 135–142
11. Floyd, R.W.: Algorithm 97: Shortest path. Commun. ACM **5**(6) (1962) 345
12. Yannakakis, M.: Computing the minimum fill-in is np-complete. SIAM Journal on Algebraic Discrete Methods **2**(1) (1981) 77–79
13. Lengauer, T., Tarjan, R.E.: A fast algorithm for finding dominators in a flowgraph. ACM Trans. Program. Lang. Syst. **1**(1) (January 1979) 121–141
14. Narodytska, N., Bacchus, F.: Maximum satisfiability using core-guided maxsat resolution. In: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada. (2014) 2717–2723