SAT-Based Approaches to Treewidth Computation: An Evaluation

Jeremias Berg and Matti Järvisalo
HIIT & Department of Computer Science, University of Helsinki, Finland

***Abstract*—Treewidth is an important structural property of graphs, tightly connected to computational tractability in eg various constraint satisfaction formalisms such as constraint programming, Boolean satisfiability, and answer set programming, as well as probabilistic inference, for instance. An obstacle to harnessing treewidth as a way to efficiently solving bounded treewidth instances of NP-hard problems is that deciding treewidth, and hence computing an optimal tree-decomposition, is in itself an NP-complete problem. In this paper, we study the applicability of Boolean satisfiability (SAT) based approaches to determining the treewidths of graphs, and at the same time obtaining an associated optimal tree-decomposition. Extending earlier studies, we evaluate various SAT and MaxSAT based strategies for treewidth computation, and compare these approaches to practical dedicated exact algorithms for the problem.**

I. Introduction

Treewidth is a fundamental and important graph property, which intuitively characterizes "how far" an undirected graph is from being a tree. In general, the treewidth of graphs on n nodes ranges from 1 (for trees) to $n-1$ (including the complete graph K_n). From the computational perspective, treewidth has important connections to (in)tractability. Specifically, many NP-hard problems on graphs, when restricted to input graphs of bounded treewidth, can be solved in polynomial time via dynamic programming algorithms on *tree decompositions* through which treewidth of graphs can be defined. This is due to the fact that combinatorial explosion (exponentiality) can be confined to the structural parameter of treewidth, and hence bounded (low) treewidth implies fast computation. From the practical perspective, empirical studies in various domains [1], [2], [3], [4] suggest that treewidth of practical instances is indeed often small.

In addition to various NP-hard combinatorial problems defined directly over graphs [5], [6], [7], [8], [9], this also holds eg for constraint satisfaction problems [10], [11], [12], [13] and instances of Boolean satisfiability (SAT) the underlying incidence graphs of which have bounded treewidth [14], [15]. In the context of machine learning bounded treewidth enables polynomial-time inference over probabilistic graphical models such as Bayesian networks via the dynamic programming style junction tree algorithm for belief propagation [16], which has motivated recent work on SAT-based approaches to learning bounded treewidth Bayesian network structures [17]. Similarly, dynamic programming over tree-decompositions has been recently proposed as a way of developing novel types of solvers in the important non-monotonic declarative programming paradigm of Answer Set Programming (ASP) [18].

An obstacle to harnessing treewidth as a way to efficiently solving bounded treewidth instances of NP-hard problems is that the typical dynamic programming algorithms working on tree-decompositions rely on the optimality of the tree-decompositions, i.e., tree-decompositions which are directly associated with the treewidth of the graph underlying the problem instance. Finding such optimal tree-decompositions, in turn, is NP-hard; in fact, merely determining if a given undirected graph has treewidth at most k is an NP-complete problem [19]. While this problem has been the subject of various theoretical studies [20], [21], including approximation techniques [22], [23], only few practical exact algorithms have been proposed [24], [25].

In this paper, we study the applicability of Boolean satisfiability (SAT) based approaches to determining the treewidths of graphs, and at the same time obtaining an associated optimal tree-decomposition. Extending previous work on SAT-based treewidth computation [26], our contributions are the following.

- We describe various SAT-based strategies and slight encoding variants for the problem, including both strategies which directly employ a SAT solver in an incremental fashion, as well as directly making a single Maximum satisfiability (MaxSAT) solver invocation.
- Compared to earlier work [26], we perform a much more extensive evaluation of these SAT-based approaches, including using multiple incremental SAT strategies and multiple different types of MaxSAT solvers.
- We provide a comparison with other dedicated exact algorithmic approaches proposed for treewidth computation—including dynamic programming and branch-and-bound algorithms—which is missing from earlier work.

The results show that the SAT-based approach is in cases very competitive with the dedicated algorithms, being able to provide solution in cases when the dedicated algorithms fail due to memory problems. Furthermore, both incrementality (in the pure SAT-based strategies) and the MaxSAT-based strategy show clear improvements over the baseline SAT-based approach presented in [26].

The rest of this paper is organized as follow. We start by preliminaries on treewidth and tree-decompositions (Section II) and SAT and MaxSAT(Section III). We then continue by a detailed description of the SAT-based encodings and solving strategies to determining treewidth (Section IV). This is followed by the description and results of an empirical evaluation of the SAT-based approaches and their comparison with implementations of previously proposed dedicated exact algorithms to determining treewidth (Section V).

II. TREEWIDTH

The treewidth of an undirected graph G is often defined in terms of the *tree-decompositions* of G.

Definition A tree-decomposition of an undirected graph $G = (V, E)$ is a tree T over a set $\{V_1, \ldots, V_m\}$ of nodes, where $V_i \subseteq V$, with the following properties.

1) $\cup_{i=1}^{m} V_i = V$.
2) If $\{u, v\} \in E$, then $u, v \in V_i$ for some $i \in \{1, \ldots, m\}$.
3) For all $i, j, k \in \{1, \ldots, m\}$, the following holds: if V_j is on the (unique) path from V_i to V_k in T, then $V_i \cap V_k \subseteq V_j$.

The width of a tree-decomposition is $\max_{i=1}^{m} |V_i| - 1$.

Definition The treewidth $tw(G)$ of an undirected graph G is the minimum width over all tree-decompositions of G.

It is well-known that, for any undirected graph $G = (V, E)$, any linear ordering of the nodes V of G defines a tree-decomposition of G, and that there is always an "optimal" linear ordering of V defining an *optimal* tree-decomposition, i.e., a tree-decomposition of width $tw(G)$ [27], [28]. Furthermore, without needing to explicitly construct the corresponding optimal tree-decomposition, the treewidth of G can be determined based on an optimal linear ordering $\prec$ of V. A node $v_i \in V$ is a *predecessor* of $v_j \in V$ under $\prec$ if $i \prec j$ and $\{v_i, v_j\} \in E$; v_i is a *successor* of v_j under $\prec$ if $j \prec i$ and $\{v_i, v_j\} \in E$. Given a linear ordering $\prec$ of V, the width of the corresponding tree-decomposition is determined by applying the following *triangulation* procedure under $\prec$ to G.

Repeat the following as long as new edges can be added to E:

For each pair v_i, v_j of nodes with a common predecessor, add the edge $\{v_i, v_j\}$ to E.

We denote the resulting edge-relation by $\Delta(E, \prec)$, defining the *triangulation* $\Delta(G, \prec) = (V, \Delta(E, \prec))$ of G under $\prec$. Orienting the edges of $\Delta(G, \prec)$ according to $\prec$ gives the directed edge-relation

$$\vec{\Delta}(E, \prec) = \{(v_i, v_j) \mid \{v_i, v_j\} \in \Delta(E, \prec),\ i \prec j\}$$

defining the *ordered graph* $\vec{\Delta}(G, \prec) = (V, \vec{\Delta}(E, \prec))$ of G under $\prec$. Now, the width of the tree-decomposition defined by $\prec$ is

$$\max_{v_i \in V} |\{(v_i, v_j) \in \vec{\Delta}(E, \prec)\}|, \tag{1}$$

i.e., the maximum number of successors over all nodes in $\Delta(E, \prec)$. The treewidth $tw(G)$ of G is then

$$\min_{\prec} \max_{v_i \in V} |\{(v_i, v_j) \in \vec{\Delta}(E, \prec)\}| \tag{2}$$

over all linear orderings $\prec$ of the nodes V of G. In other words, an alternative characterization of the treewidth of G is the smallest value w for which there exists a linear order $\prec$ of V s.t.

$$\max_{v_i \in V} |\{(v_i, v_j) \in \vec{\Delta}(E, \prec)\}| \leq w.$$

The SAT-based approaches for determining $tw(G)$ are all based on this characterization.

Example Consider the the graph G in Figure 1(a). A tree-decomposition of G is shown in Figure 1(b). This tree-decomposition corresponds to using the linear ordering $v_6 \prec v_2 \prec v_4 \prec v_1 \prec v_3 \prec v_5$, under which the triangulation procedure produces the triangulation $\Delta(E, \prec)$ of G shown in Figure 1(c), and further the ordered graph $\vec{\Delta}(G, \prec)$ of G under $\prec$ shown in Figure 1(d). For this ordering, Equation 1 evaluates to 2, and hence the treewidth of G is at most 2. It can be checked that this $\prec$ actually gives an optimal tree-decomposition (shown in Figure 1(b)) of G. A simple motivation for this claim is that G is not a tree, and hence $tw(G) > 1$, which implies G has treewidth 2.

III. SAT AND MAXSAT

For a Boolean variable x, there are two literals, x and $\neg x$. A clause is a disjunction ($\vee$, logical OR) of literals. A truth assignment is a function from Boolean variables to $\{0, 1\}$. A clause C is satisfied by a truth assignment τ ($\tau(C) = 1$) if $\tau(x) = 1$ for a literal x in C, or $\tau(x) = 0$ for a literal $\neg x$ in C. A set F of clauses, i.e., a CNF formula, is satisfiable if there is an assignment τ satisfying all clauses in F ($\tau(F) = 1$), and unsatisfiable ($\tau(F) = 0$ for every assignment τ) otherwise. The well-known NP-complete Boolean satisfiability (SAT) problem asks whether a given CNF formula is satisfiable.

An instance $F = (F_h, F_s)$ of the *partial MaxSAT* problem consists of two sets of clauses: a set F_h of *hard* clauses and a set F_s of *soft* clauses. Any truth assignment τ that satisfies F_h is a *solution* to F. The *cost* of a solution τ to F is

$$\text{COST}(F, \tau) = \sum_{C \in F_s} (1 - \tau(C)),$$

i.e., the number of soft clauses not satisfied by τ. A solution τ is (globally) *optimal* for F if $\text{COST}(F, \tau) \leq \text{COST}(F, \tau')$ holds for any solution τ' to F. The cost of the optimal solutions of F is denoted by $\text{OPT}(F)$. Given a partial MaxSAT instance F, the partial MaxSAT problem asks to find an optimal solution to F. From here on, we refer to partial MaxSAT instances simply as MaxSAT instances.

IV. SAT-BASED TREEWIDTH COMPUTATION

In this section, we outline SAT-based approaches to determining the treewidths of given undirected graphs. In particular, based on an underlying SAT encoding of treewidth, we detail several SAT-based strategies, allowing for either directly using a SAT solver iteratively to determine treewidth, or determining treewidth with a single call to a MaxSAT solver. In the experiments presented in this paper, we compare these strategies on standard treewidth benchmarks, as well as compare the approach to dedicated exact algorithms for determining treewidth.

A. Base Encoding

For enforcing the treewidth bound on the input graph $G = (V, E)$, we follow—with modifications—a SAT encoding of treewidth in undirected graphs presented in [26]. Specifically, we do not encode the construction of a tree-decomposition of

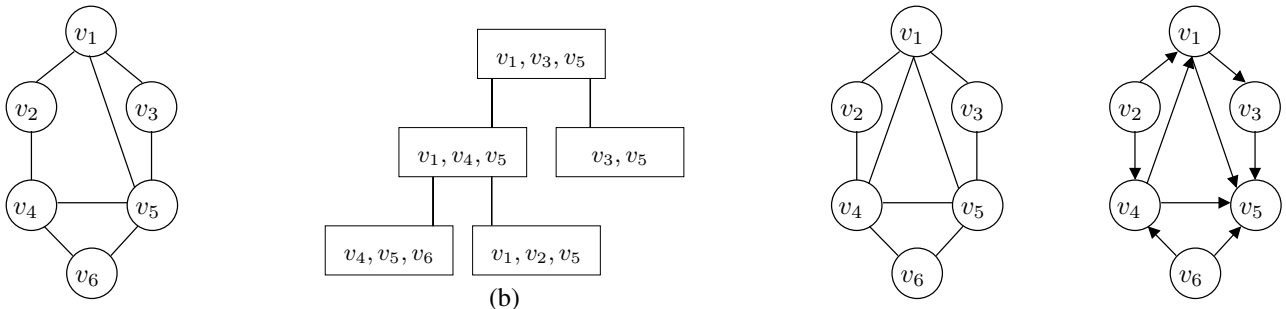

Fig. 1. Example: (a) An example undirected graph G; (b) The tree-decomposition associated with the linear ordering $v_6 \prec v_2 \prec v_4 \prec v_1 \prec v_3 \prec v_5$; (c) the triangulation $\Delta(G, \prec)$ of G under $\prec$; (d) the ordered graph $\vec{\Delta}(G, \prec)$.

G explicitly. Instead, our encoding enforces that, if there is a linear ordering $\prec$ of V under which the maximum number of successors over all nodes in the ordered graph of G is at most some constant w, then the treewidth of G is at most w.

We start by outlining a base encoding shared by all of the SAT-based treewidth computation variants we consider. Given a graph $G = (V, E)$ over N nodes as input, the base encoding consists of (1) representing linear orderings of the nodes V, and, for a given linear ordering, (2) representing the ordered graph of G.

The Boolean variables used are the following:

- ord_{ij} for all $i, j = 1..N$ such that $i < j$ represent a linear ordering $\prec$ of the nodes of G: $ord_{ij} = 1$ iff $v_i \prec v_j$.
- O_{ij} for all $i, j = 1..N$ such that $i \neq j$ represent the ordered graph $\vec{\Delta}(G, ord) : O_{ij} = 1$ iff the ordered graph of G under $\prec$ contains the edge (v_i, v_j).

1) Encoding Linear Orderings: The choice of a linear ordering of V is represented by the ord_{ij} variables. For notational convenience, let

$$ord^*_{ij} = \begin{cases} ord_{ij} & \text{if } i < j \\ \neg ord_{ji} & \text{else} \end{cases}.$$

Transitivity of linear orderings is enforced in the encoding by stating for all distinct $i, j, k = 1..N$

$$ord^*_{ij} \wedge ord^*_{jk} \rightarrow ord^*_{ik}. \tag{3}$$

2) Encoding the Ordered Graphs: The O_{ij} variables, encoding the ordered graph induced by a linear ordering $\prec$ given by an assignment over ord_{ij}, are declared as follows.

If the graph contains an edge $\{v_i, v_j\}$, then the triangulation of the graph also contains the edge $\{v_i, v_j\}$, and hence the ordered graph contains either the edge (v_i, v_j) or the edge (v_j, v_i). This is enforced by

$$(O_{ij} \vee O_{ji}) \tag{4}$$

for all $i < j$ for which $\{v_i, v_j\} \in E$.

If nodes v_i and v_j have a common predecessor in the graph, then the triangulation of the graph contains the edge $\{v_i, v_j\}$, and hence the ordered graph contains either the edge (v_i, v_j) or the edge (v_j, v_i). This is enforced for all distinct $i, j, k = 1..N$ by

$$(O_{ki} \wedge O_{kj}) \rightarrow (O_{ij} \vee O_{ji}). \tag{5}$$

Finally, in both Eqs. 4 and 5, the choice of which of the edges (v_i, v_j) or (v_j, v_i) occur in the ordered graph depends on the linear ordering $\prec$. Essentially, O_{ij} must be consistent with ord_{ij} in that, if $v_i \prec v_j$, then the edge (v_j, v_i) does not occur in the ordered graph under ord:

$$ord^*_{ij} \rightarrow \neg O_{ji}. \tag{6}$$

Equations 3–6 are sufficient for describing the ordered graph of G under $\prec$ as constraints. However, our encoding also includes domain specific redundant clauses that empirically were found to decrease the time required for solving the resulting instances of SAT and MaxSAT. The redundant clauses are based on the observation that the ordered graph of G is simple; whenever there is an edge (v_i, v_j), there cannot be the edge (v_j, v_i). This corresponds to enforcing for all distinct i and j the clause

$$\neg O_{ji} \vee \neg O_{ij}. \tag{7}$$

From now on, we refer to the conjunction of the clausal representation of the formulas given in Equations 3–7 as the *base encoding*. For a given graph G we denote the SAT instance created by the base encoding by $\mathcal{F}_{\text{base}}(G)$.

B. Deciding Treewidth

Recall that the treewidth of the tree-decomposition corresponding to a linear ordering $\prec$ specified by the ord_{ij} variables is $\max_{v_i \in V} |\{\{v_i, v_j\} \in \Delta(E, \prec) \ : \ i \prec j\}|$, where $\Delta(E, \prec)$ is the edge-relation of the triangulated graph. The variable O_{ij} represents the fact that the ordered graph of G under the linear ordering $\prec$ contains the edge (v_i, v_j). It follows that enforcing the cardinality constraint

$$\mathcal{C}_w(i) = \sum_{j \neq i} O_{ij} \leq w \tag{8}$$

for each $i = 1..N$ is equivalent to the requirement

$$\max_{v_i \in V} |\{\{v_i, v_j\} \in E \mid i \prec j\}| \leq w.$$

As the treewidth of G is the minimum over all linear orderings of V, encoding Equation 8 for all $i = 1..N$ as clauses, together with the base encoding, results in the SAT instance

$$\mathcal{F}_{\text{base}}(G) \wedge \bigwedge_{i=1}^{N} \mathcal{C}_w(i)$$

which is satisfiable if and only if $tw(G) \leq w$. We note that whenever the above formula is satisfiable, the variables ord_{ij} represent a linear ordering of the nodes from which a tree-decomposition of width at most w is easily constructed [28].

Based on the above the treewidth of G can hence be determined by several independent SAT solver calls in order to find the smallest value $w \in \{1, \ldots, N-1\}$ for which $\mathcal{F}_{\text{base}}(G) \wedge \bigwedge_{i=1}^{N} \mathcal{C}_w(i)$ is satisfiable. This approach for determining the treewidth of a graph is possible with any clausal encoding of Equation 8, however specific choices of the encoding allow alternative approaches as well.

C. Enabling Incremental SAT Solving

In order to enable incremental SAT solving we employ a compact clausal encoding of Equation 8 based on so-called *cardinality networks* [29], [30]. Given a set $A_i = \{O_{ij} \mid j \neq i\}$ the cardinality network encoding $C(A_i)$ produces a clausal definition of $|A_i|-1 = N-1$ auxiliary variables $y_1^i, \ldots, y_{N-1}^i$ which indicate how many of the variables in A_i are set to true. More precisely: if $\tau(y_k^i) = 0$ in a solution τ, then at most $k-1$ of the variables in A_i are set to 1 by τ. It follows that for a fixed i, the clauses $C(A_i)$ and the unit clause $(\neg y_{w+1}^i)$ are equivalent to Equation 8.

Consider now the SAT instance created by the base encoding and the cardinality networks $C(A_i)$ for all $i = 1..N$:

$$\mathcal{F}'_{\text{iter}} := \mathcal{F}_{\text{base}}(G) \wedge \bigwedge_{i=1}^{N} C(A_i).$$

The requirement $tw(G) \leq w$ is equivalent to $\mathcal{F}'_{\text{iter}}$ being satisfiable under the assumptions $\tau(y_{w+1}^i) = 0$ for all $i = 1..N$. To make use of this observation, we define auxiliary variables $\mathcal{W}_i$ for $i = 0..(N-2)$ as

$$\mathcal{W}_i \leftrightarrow (\neg y_{i+1}^1 \wedge \neg y_{i+1}^2 \wedge \ldots \wedge \neg y_{i+1}^N). \tag{9}$$

Now the treewidth of G can be determined by finding the smallest w for which $\mathcal{F}'_{\text{iter}}$ together with the clauses corresponding to Equation 9 is satisfiable under the assumption $\mathcal{W}_w = 1$. As the corner case, note that if the instance is un satisfiable under the assumption $\mathcal{W}_{N-2} = 1$, we can conclude that $tw(G) = N-1$.

Also here we add domain specific knowledge to the encoding by using the fact that for any w, whenever $tw(G) \leq w$, we have $tw(G) \leq w'$ for all $w' > w$. This is added to the encoding by implications of the form

$$\mathcal{W}_i \rightarrow \mathcal{W}_{i+1} \tag{10}$$

for all $i = 0..(N-3)$. We call the base encoding together with the clauses corresponding to $C(A_i)$ and Equations 9 and 10 the *iterative encoding*. For a given graph we denote an instance of the iterative encoding by $\mathcal{F}_{\text{iter}}(G)$. Instances of the iterative encoding can be solved using an incremental interface of a SAT solver, iteratively solving $\mathcal{F}_{\text{iter}}(G)$ under different assumptions, allowing the solver to retain information between iterations.

D. Using MaxSAT

As discussed above and declared be Eq. 10, whenever $\tau(\mathcal{W}_i) = 1$ for a solution τ to $\mathcal{F}_{\text{iter}}(G)$, $\tau(\mathcal{W}_{i'})$ can be set to 1 for all $i' > i$. Hence an alternative characterization of determining the treewidth of G is to find a solution τ which satisfies all clauses in $\mathcal{F}_{\text{iter}}(G)$ and minimizes the number of indices i for which $\tau(\mathcal{W}_i) = 0$. This characterization has a natural interpretation as a MaxSAT instance

$$\mathcal{F}_{\max}(G) = (F_h, F_s)$$

where $F_h = \mathcal{F}_{\text{iter}}(G)$ and $F_s = \{(\mathcal{W}_i) \mid i = 0..(N-2)\}$.

For some intuition, note that whenever $tw(G) = w$, any solution τ to $\mathcal{F}_{\max}(G)$ has to assign all of the variables $\mathcal{W}_0, \ldots, \mathcal{W}_{w-1}$ to 0 and can assign all of the variables $\mathcal{W}_w, \ldots, \mathcal{W}_{N-1}$ to 1. Each $\mathcal{W}_i$ variable assigned to 0 corresponds to an unsatisfied soft unit clause, and hence

$$\text{OPT}(\mathcal{F}_{\max}(G)) = w = tw(G).$$

E. The Original SAT Encoding of [26]

The base encoding and its iterative extension are similar to the SAT encoding for deciding an upper bound on the treewidth proposed in [26]. The most important difference between the two is the choice of cardinality encoding for the constraint described Equation 8. In the original SAT encoding, Equation 8 is encoded as clauses using a so called *improved sequential counter*. For our work the important difference between the encodings to note is that this encoding is very difficult, if not impossible, to use efficiently in an iterative setting. Hence the only viable approach of using the original encoding for determining the treewidth of a given graph is individually solving several SAT instances with different upper bounds. We refer the reader to [26] for the specifics of the improved sequential counter.

Our base encoding is also slightly more compact than the original SAT encoding in [26]. The original encoding does not include the clauses corresponding to Eqs. 4, 6 or 7. Instead, it includes the clausal representation of $ord_{ij}^* \rightarrow O_{ij}$ and $ord_{ji}^* \rightarrow O_{ji}$ for each edge in the input graph G, in order to ensure that all of these edges are included in the ordered graph. In order to describe the induced edges in the ordered graph, the encoding includes constraints of the form

$$(O_{ki} \wedge O_{kj} \wedge ord_{ij}^*) \rightarrow O_{ij}$$

and

$$(O_{ki} \wedge O_{kj} \wedge ord_{ji}^*) \rightarrow O_{ji}.$$

F. SAT-based Approaches to Computing Treewidth

In the following, we empirically compare three different SAT-based approaches for determining the treewidth of a given graph G with N nodes.

1) The original strategy proposed in [26] in which we individually solve at most $N-2$ different SAT instances using the original encoding of [26]. We report the sum of the SAT

solving times required to determine $tw(G)$ using two different search strategies:

- **Orig-U:** Start with the lower bound 1, and increase it while the instance is unsatisfiable.
- **Orig-D:** Start with the upper bound $N - 2$, and decrease it while the instance is satisfiable.

2) An iterative approach in which the incremental interface of a SAT solver is used in order to find the smallest w for which $\mathcal{F}_{\text{iter}}(G)$ is satisfiable under the assumption $\mathcal{W}_w = 1$. We consider three different search strategies:

- **Iter-U:** Bottom-up linear search, starting from $w = 1$ and increasing until $\mathcal{F}_{\text{iter}}(G)$ is satisfiable.
- **Iter-D:** Top-down linear search, starting from $w = N - 2$ and decreasing until $\mathcal{F}_{\text{iter}}(G)$ is satisfiable.
- **Iter-Bin:** Binary search, updating the upper bound whenever $\mathcal{F}_{\text{iter}}(G)$ is satisfiable under the current assumptions and the lower bound whenever its unsatisfiable. The search terminates when the upper bound and lower bound are equal.

3) Using MaxSAT solvers to find an optimal solution to $\mathcal{F}_{\max}(G)$. We call the resulting approach **MaxSAT**.

V. Experiments

We provide and empirical comparison of the three different SAT-based approaches for computing the treewidth of different graphs on several standard benchmarks.

For solving the resulting SAT instances we used Minisat version 2.2 [31]. MiniSAT provides an incremental interface, allowing us to use it for both the individual SAT instances as well as the iterative approach. For the incremental approach we also used Glucose 3.0 with the optimizations for incremental use described in [32]. For the MaxSAT instances, we used WMaxSatz09 [33], MSUnCore [34], [35] bcd2 version, MaxHS [36] MaxSAT Evaluation 2013 version, and OpenWBO [37] version 1.1.1. These solvers are based on different types of algorithms. WMaxSatz is a branch-and-bound solver that maintains an upper and lower bound while searching for the optimal MaxSAT solution. MSUnCore and OpenWBO are based on using a SAT-solver as a black box in order to identify smaller unsatisfiable cores from the original MaxSAT instance. The soft clauses in each identified core are then relaxed in the working formula and the process is repeated until the optimal cost of the MaxSAT instance is identified. OpenWBO also splits the set of soft clauses into smaller subsets and searches for unsatisfiable cores separately from the different partitions. MaxHS, on the other hand, implements a hybrid SAT-MIP approach based on iteratively solving a sequence of SAT instances and extracting cores, and using the IBM CPLEX MIP solver to solve a sequence of minimum hitting set problems over the extracted cores.

As benchmarks we used several real world graph coloring benchmarks, obtained from http://www.staff.science.uu.nl/~bodla101/treewidthlib/coloring.zip. We also experimented Bayesian Networks from the UCI repository as well as the graphs used in [26], available from http://www.cs.uu.nl/research/projects/treewidthlib/. Our benchmarks range from 11 to 128 nodes, with a highest treewidth of 22. More information regarding the benchmarks can be found in Table I.

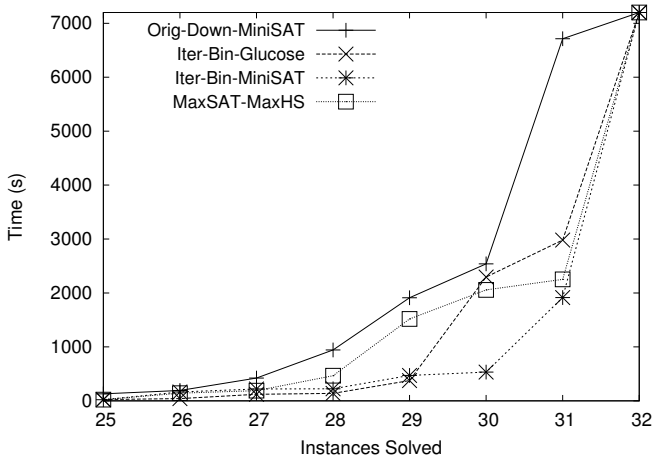

Fig. 2. Comparison of the SAT-based approaches

The experiments were performed on a cluster of 2.8-GHz Intel Xeon quad core machines with 32-GB memory and Ubuntu Linux 10.04. A timeout of 2 h (7200 seconds) and a memory limit of 30 GB were enforced on the solvers on the individual benchmark instances.

A. Results: Comparison of the SAT-based Approaches

A comparison of the running times for the different SAT-based approaches is presented in Table I. For the original non-incremental approach used in [26], the bottom-up search strategy **Orig-U** performs better than the top-down strategy **Orig-U**. For the incremental iterative strategies using MiniSAT, the binary search strategy **Iter-Bin** outperforms the bottom-up and top-down strategies **Iter-U** and **Iter-D**. We also report the running times using Glucose with its improved incremental engine [32] under the binary search strategy, **Iter-Bin-G**. While many of the easier instances where solved faster with Glucose, Glucose used notably more time on some of the harder benchmarks such as Eil51.

Figure 2 gives the number of solved instances within different time limits. Clearly, the original SAT encoding and the approach of individually solving several SAT instances is not competitive to the other two approaches considered. The same can be seen in the scatter plots in Figure 3 (left, middle). The results are to be expected. A modern SAT solver and MaxSAT solver is able to memorize and use information it learns in earlier iterations in order to speed up solving in the later SAT iterations. As seen in Figure 3 (right), the performance difference between iterative SAT and MaxSAT is small and the results inconclusive.

B. Results: Comparison with Specialized Algorithms

We compare the SAT-based approaches to the dynamic programming algorithm (DP) of [25] and the branch-and-bound algorithm QuickBB of [24]. The implementation of

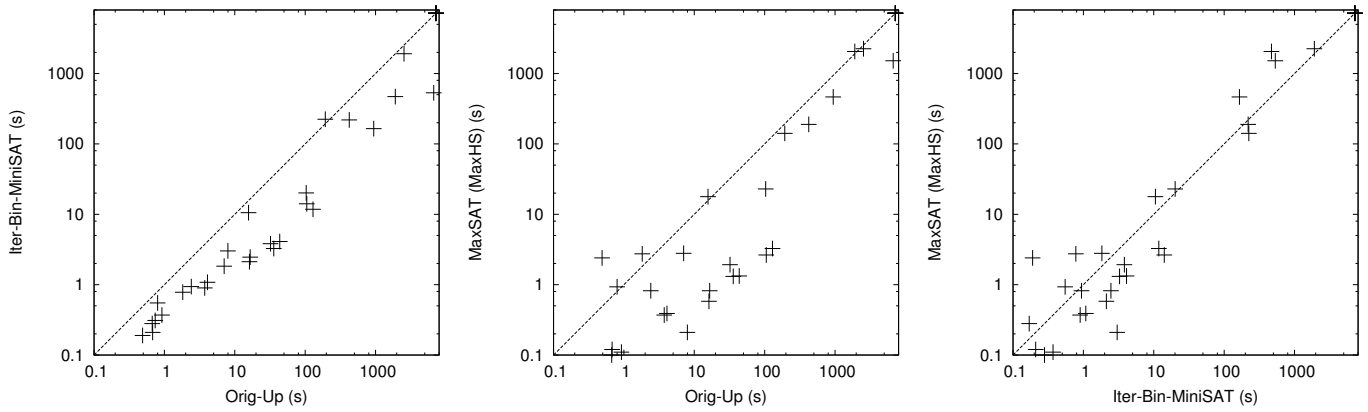

Fig. 3. Pairwise comparison of the SAT-based approaches. Left: Incremental binary search vs independent bottom-up linear search (MiniSAT); middle: MaxSAT (MaxHS) vs independent bottom-up linear search using SAT (MiniSAT); right: MaxSAT (MaxHS) vs incremental binary search using SAT (MiniSAT)

the DP algorithm was obtained from its authors. The implementation of QuickBB was obtained from http://graphmod.ics.uci.edu/group/quickbb. We next briefly describe both of these algorithms.

Given a graph $G = (V, E)$ and a subset $S \subset V$ we denote by $G[S]$ the subgraph of G consisting of the nodes in S and all the edges $\{x, y\}$ such that $x \in S$ and $y \in S$. The DP algorithm is based on the observation that the treewidth $G[S]$ can be effectively determined given the treewidth of $G[S \setminus \{x\}]$ for all $x \in S$. The basic form of the algorithm iterates over increasing sizes of subsets S of V and determines the treewidth of $G[S]$, the treewidth of G being equal to the treewidth of $G[V]$. The performance of the algorithm is improved by using a number of heuristics [25]. However, the space requirement of the algorithm is still exponential in the number of nodes.

QuickBB searches over perfect elimination orderings of the nodes of G. The algorithm is based on fact that the width of an ordering $o\colon S \to \{1..|S|\}$ of some subset $S \subset V$ is a lower bound for the width of any ordering obtained when extending o to include the whole of V in a way that every node in S comes before any node in $V \setminus S$. QuickBB starts by calculating a lower and upper bound of $tw(G)$ using heuristics. If the bounds are equal, the problem is solved. Else QuickBB starts branch-and-bound search, iteratively constructing an ordering of the nodes. At each step a heuristic is used for calculating the lower bound of the partial order currently under consideration. Whenever the lower bound is greater than the known upper bound for the treewidth, that branch in the search is terminated. Else the search branches, each branch adding one of the remaning nodes last to the current ordering. Whenever all the nodes have been added to the ordering under consideration, the algorithm checks if the width of the ordering is lower than the current upper bound and updates the upper bound if it is. QuickBB also uses a number of additional heuristics in order to prune the search space and limit the number of generated branches; we refer the reader to [24] for details.

The results of the comparison are reported in Table I. We observe that dynamic programming approach is competitive with the SAT-based approaches only for the smallest datasets, more importantly, the dynamic programming most often runs out of memory, a conclusion similar to [25]. Memory outs can be argued to be even more critical than timeouts as they imply that the solver will never find the optimal solution, regardless of how much computation time is given. Compared to the DP, QuickBB performs notably better, often finding the treewidth of the input graph within seconds. However, we also observe 4 instances that were solvable by all of the SAT-based approaches but on which QuickBB ran out of memory. Indeed, based on the results it appears that the incremental SAT-based and the specialized exact algorithm implementations, especially QuickBB, offer complementary approaches to exact treewidth computation.

C. Obtaining Anytime Bounds via SAT

Finally, we shortly mention that the SAT-based approaches can naturally provide upper and/or lower bounds—depending on the search strategy used—for treewidth during search. In fact, we observed that for most of the benchmark instances, only few of the SAT solver calls were difficult for the solvers, meaning that relatively tight bounds can be found in cases relatively fast. To exemplify this, the evolution of the upper and lower bounds on two benchmark graphs are illustrated in Figure 4, using Glucose and the binary search strategy. As future work, it would be interesting to compare the bounds

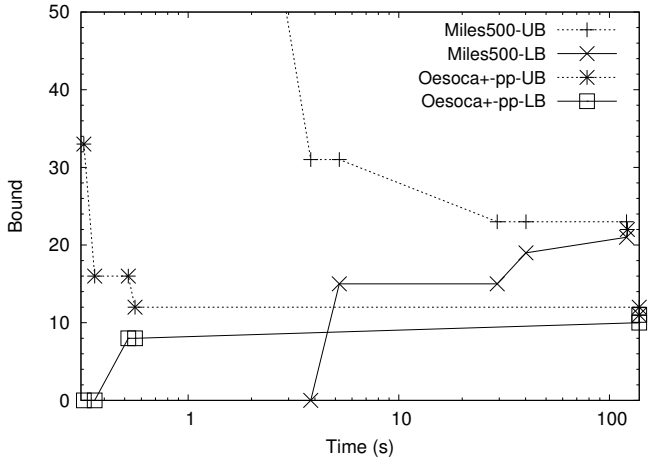

Fig. 4. Evolution of the upper and lower bound over time for two datasets. Solved by Glucose and binary search.

achieved by SAT-based approaches to bounds that can be obtained via specialized approximation algorithms for treewidth.

VI. Conclusions

We presented and experimentally evaluated different SAT-based methods for determining (computing) the treewidth of a given graph. We extended a previously proposed SAT encoding to enable iterative SAT and MaxSAT, as well as proposed modifications to the encoding. Compared to earlier work, we reported on a notably broader experimental evaluation of the SAT-based based approaches. The results demonstrate that both iterative SAT and MaxSAT outperform the "naive" SAT-based method earlier proposed. Further, we experimentally compared the SAT-based approaches to previously proposed exact treewidth computation algorithms based on dynamic programming and branch-and-bound search. The SAT-based approach is much more memory efficient than both the dynamic programming and branch-and-bound approaches. In whole, the results suggest that SAT / MaxSAT is a viable complementary approach to exactly determining the treewidth of moderately sized graphs.

Acknowledgements. Work supported by Academy of Finland, grants 251170 (Finnish Centre of Excellence in Computational Inference Research COIN) and 276412, and Finnish Funding Agency for Technology and Innovation (project D2I).

References

[1] R. Agarwal, P. B. Godfrey, and S. Har-Peled, "Approximate distance queries and compact routing in sparse graphs," in *INFOCOM*. IEEE, 2011, pp. 1754–1762.

[2] M. Thorup, "All structured programs have small tree-width and good register allocation," *Information and Computation*, pp. 159–181, 1998.

[3] J. Gramm, A. Nickelsen, and T. Tantau, "Fixed-parameter algorithms in phylogenetics," *Computer Journal*, vol. 51, no. 1, pp. 79–101, 2008.

[4] X. Huang and J. Lai, "Parameterized graph problems in computational biology," in *Proc. IMSCCS*. IEEE, 2007, pp. 129–132.

[5] R. B. Borie, R. G. Parker, and C. A. Tovey, "Solving problems on recursively constructed graphs," *ACM Computing Surveys*, vol. 41, no. 1, 2008.

[6] B. Aspvall, J. A. Telle, and A. Proskurowski, "Memory requirements for table computations in partial k-tree algorithms," *Algorithmica*, vol. 27, no. 3, pp. 382–394, 2000.

[7] T. Wimer, S. Hedetniemi, and R. Laskar, "A methodology for constructing linear graph algorithms," *Congressus Numerantium*, pp. 43–60, 1985.

[8] S. Arnborg and A. Proskurowski, "Linear time algorithms for NP-hard problems restricted to partial k-trees," *Discrete Applied Mathematics*, vol. 23, no. 1, pp. 11–24, 1989.

[9] M. W. Bern, E. L. Lawler, and A. L. Wong, "Linear-time computation of optimal subgraphs of decomposable graphs," *Journal of Algorithms*, vol. 8, no. 2, pp. 216–235, 1987.

[10] A. M. C. A. Koster, S. P. M. van Hoesel, and A. W. J. Kolen, "Solving partial constraint satisfaction problems with tree decomposition," *Networks*, vol. 40, no. 3, pp. 170–180, 2002.

[11] G. Gottlob, G. Greco, and F. Scarcello, "Tractable optimization problems through hypergraph-based structural restrictions," in *Proc. ICALP*, ser. LNCS, vol. 5556. Springer, 2009, pp. 16–30.

[12] V. Dalmau, P. G. Kolaitis, and M. Y. Vardi, "Constraint satisfaction, bounded treewidth, and finite-variable logics," in *Proc. CP*, ser. LNCS, vol. 2470. Springer, 2002, pp. 310–326.

[13] H. Chen, "Quantified constraint satisfaction and bounded treewidth," in *Proc. ECAI*. IOS Press, 2004, pp. 161–165.

[14] E. Fischer, J. A. Makowsky, and E. V. Ravve, "Counting truth assignments of formulas of bounded tree-width or clique-width," *Discrete Applied Mathematics*, vol. 156, no. 4, pp. 511–529, 2008.

[15] F. Slivovsky and S. Szeider, "Model counting for formulas of bounded clique-width," in *Proc. ISAAC*, ser. LNCS, vol. 8283. Springer, 2013, pp. 677–687.

[16] S. L. Lauritzen and D. J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 50, no. 2, pp. 157–224, 1988.

[17] J. Berg, M. Järvisalo, and B. Malone, "Learning optimal bounded treewidth bayesian networks via maximum satisfiability," in *Proc. AISTATS*, ser. JMLR Proceedings, vol. 33. JMLR.org, 2014, pp. 86–95.

[18] B. Bliem, M. Morak, and S. Woltran, "D-FLAT: Declarative problem solving using tree decompositions and answer-set programming," *Theory and Practice of Logic Programming*, vol. 12, no. 4-5, pp. 445–464, 2012.

[19] S. Arnborg, D. G. Corneil, and A. Proskurowski, "Complexity of finding embeddings in a k-tree," *SIAM J. Algebraic Discrete Methods*, vol. 8, no. 2, pp. 277–284, Apr. 1987.

[20] H. L. Bodlaender and A. M. C. A. Koster, "Treewidth computations ii. lower bounds," *Information and Computation*, vol. 209, no. 7, pp. 1103–1119, 2011.

[21] ——, "Treewidth computations i. upper bounds," *Information and Computation*, vol. 208, no. 3, pp. 259–275, 2010.

[22] E. Amir, "Efficient approximation for triangulation of minimum treewidth," in *Proc. UAI*. Morgan Kaufmann, 2001, pp. 7–15.

[23] A. Becker and D. Geiger, "A sufficiently fast algorithm for finding close to optimal junction trees," in *Proc. UAI*. Morgan Kaufmann, 1996, pp. 81–89.

[24] V. Gogate and R. Dechter, "A complete anytime algorithm for treewidth," in *Proc. UAI*. AUAI Press, 2004, pp. 201–208.

[25] H. L. Bodlaender, F. V. Fomin, A. M. C. A. Koster, D. Kratsch, and D. M. Thilikos, "On exact algorithms for treewidth," *ACM Transactions on Algorithms*, vol. 9, no. 1, p. 12, 2012.

[26] M. Samer and H. Veith, "Encoding treewidth into SAT," in *Proc. SAT*, ser. LNCS. Springer, 2009, vol. 5584, pp. 45–50.

[27] R. Dechter, "Bucket elimination: A unifying framework for reasoning," *Artificial Intelligence*, vol. 113, no. 1-2, pp. 41–85, 1999.

[28] H. L. Bodlaender, "Discovering treewidth," in *Proc. SOFSEM*, ser. LNCS, vol. 3381. Springer, 2005, pp. 1–16.

[29] R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell, "Cardinality networks: a theoretical and empirical study," *Constraints*, vol. 16, no. 2, pp. 195–221, 2011.

[30] I. Abío, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell, "A parametric approach for smaller and better encodings of cardinality constraints," in *Proc. CP*, ser. LNCS, vol. 8124. Springer, 2013, pp. 80–96.

[31] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Proc. SAT*, ser. LNCS, vol. 2919. Springer, 2004, pp. 502–518.

[32] G. Audemard, J. Lagniez, and L. Simon, "Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction," in *Proc. SAT*, ser. LNCS, vol. 7962. Springer, 2013, pp. 309–317.

[33] C. M. Li, F. Manyá, N. O. Mohamedou, and J. Planes, "Exploiting cycle structures in Max-SAT," in *Proc. SAT*, ser. LNCS, vol. 5584. Springer, 2009, pp. 467–480.

[34] F. Heras, A. Morgado, and J. Marques-Silva, "Core-guided binary search algorithms for maximum satisfiability," in *Proc. AAAI*. AAAI Press, 2011.

[35] A. Morgado, F. Heras, and J. Marques-Silva, "Improvements to core-guided binary search for MaxSAT," in *Proc. SAT*, ser. LNCS, vol. 7317. Springer, 2012, pp. 284–297.

[36] J. Davies and F. Bacchus, "Exploiting the power of MIP solvers in Maxsat," in *Proc. SAT*, ser. LNCS, vol. 7962. Springer, 2013, pp. 166–181.

[37] R. Martins, V. M. Manquinho, and I. Lynce, "On partitioning for maximum satisfiability," in *Proc. ECAI*. IOS Press, 2012, pp. 913–914.

TABLE I. THE RUNNING TIMES OF THE DIFFERENT SAT-BASED APPROACHES AND SPECIALIZED ALGORITHMS FOR DETERMINING TREEWIDTH. FOR EACH GRAPH, THE FASTEST SEARCH STRATEGY/SOLVER OF EACH SAT-BASED APPROACH (SAT, INCREMENTAL SAT, MAXSAT) IS SHOWN IN ***bold italic***. THE BEST RUNNING TIMES OVER ALL METHODS (BOTH SAT-BASED AND SPECIALIZED ALGORITHMS) ARE SHOWN IN **BOLD**. **TW**: TREEWIDTH (RANGE DENOTES BEST FOUND BOUNDS). $|V|$: NUMBER OF VERTICES IN THE INPUT GRAPH. $|E|$: NUMBER OF EDGES IN THE INPUT GRAPH. TO: TIMEOUT. MEM: MEMORY OUT

				Non-incremental SAT		Incremental SAT				MaxSAT				Specialized	
Instance	**TW**	$\|V\|$	$\|E\|$	**Orig-U**	**Orig-D**	**Iter-Lin-U**	**Iter-Lin-D**	**Iter-Bin**	**Iter-Bin-G**	**MaxHS**	**MSUnCore**	**OpenWBO**	**WMaxSatz**	**QuickBB**	**DP**
Myciel 3	5	11	20	**<0.1**	**<0.1**	**<0.1**	**<0.1**	**<0.1**	**<0.1**	**<0.1**	**<0.1**	**<0.1**	47.57	**<0.1**	**<0.1**
Pathfinder-pp	6	12	43	**<0.1**	**<0.1**	**<0.1**	**<0.1**	**<0.1**	**<0.1**	**<0.1**	**<0.1**	**<0.1**	79.88	**<0.1**	**<0.1**
Oesoca+-pp	11	14	75	**<0.1**	**<0.1**	0.25	***0.14***	0.17	0.43	***0.28***	0.55	0.45	TO	**<0.1**	**<0.1**
Child	3	20	30	**<0.1**	0.33	**<0.1**	**<0.1**	**<0.1**	**<0.1**	**<0.1**	**<0.1**	**<0.1**	525.54	**<0.1**	MEM
Anna-pp	12	22	148	0.80	***0.69***	1.00	***0.37***	0.55	1.04	0.93	2.13	***0.74***	TO	**<0.1**	**<0.1**
Myciel 4	10	23	71	15.69	***13.06***	***7.62***	14.14	10.53	23.71	***17.80***	35.79	18.62	TO	**<0.1**	4.95
Queen 5	18	25	160	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	0.90	**<0.1**
Barley-pp	7	26	78	***0.49***	1.27	3.12	0.23	**0.19**	0.25	2.40	***0.44***	2.11	TO	0.81	221.39
David-pp	13	29	191	***1.82***	2.07	2.76	1.02	**0.78**	0.94	2.74	4.48	***2.10***	TO	0.94	52.34
Ship-ship-pp	8	30	77	***192.90***	194.37	79.34	**39.93**	224.09	41.11	141.20	373.81	***43.49***	TO	MEM	6291.38
Water	8	32	123	***2.41***	4.50	0.81	***0.66***	0.94	1.01	***0.82***	2.00	0.70	TO	**0.20**	MEM
Mildew	4	35	80	***0.68***	6.39	0.19	0.25	0.21	**<0.1**	***0.12***	0.58	0.14	TO	**<0.1**	MEM
Queen 6	25	36	290	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	31.83	**23.51**
Alarm	4	37	65	***0.74***	7.88	0.23	0.27	0.31	**<0.1**	**<0.1**	0.64	0.14	TO	**<0.1**	MEM
Oesoca	3	39	72	***0.67***	10.60	0.28	0.34	0.28	**<0.1**	**<0.10**	0.74	0.16	TO	**<0.1**	MEM
Oesoca-42	3	42	67	***0.92***	15.45	0.37	0.46	0.37	**< 0.1**	**<0.10**	0.95	0.20	TO	MEM	MEM
Myciel 5	[15, 19]	47	236	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	MEM	MEM
Barley	7	48	126	***7.98***	30.78	***0.26***	2.58	3.02	1.80	**0.2**	4.58	0.34	TO	0.86	MEM
Barley2	7	48	126	***7.06***	30.32	3.28	2.36	***1.83***	1.93	2.79	6.84	***1.89***	TO	0.97	**<0.1**
Mainuk	7	48	198	***3.74***	26.98	0.84	0.78	0.90	***0.20***	***0.37***	2.32	0.59	TO	**<0.1**	MEM
Queen 7	[19, 35]	49	476	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	MEM
Eil51	8	51	140	***6713.35***	6762.93	653.03	933.45	**532.94**	2981.46	***1517.53***	4919.09	2826.11	TO	MEM	MEM
Hailfinder	4	56	99	***4.09***	64.44	1.04	1.25	1.08	***0.20***	***0.39***	2.72	0.56	MEM	**<0.1**	MEM
Cellar09-pp	7	67	165	***16.15***	170.40	2.19	2.42	2.12	***0.32***	***0.58***	6.32	1.28	MEM	**<0.1**	MEM
Oesoca+	11	67	208	***422.79***	566.48	167.17	259.73	219.32	***138.51***	189.53	380.08	***153.10***	MEM	**<0.1**	MEM
1c75	[22, 29]	69	683	TO	TO	TO	TO	TO	TO	TO	TO	TO	MEM	MEM	MEM
1en2	16	69	463	***1910.56***	1961.70	692.02	**332.44**	470.56	375.40	2056.78	TO	***589.90***	MEM	MEM	MEM
Hepar2	6	70	158	***16.54***	201.04	2.34	3.11	2.46	***0.47***	***0.82***	6.86	1.48	MEM	**<0.1**	MEM
1dj7	[22, 26]	73	743	TO	TO	TO	TO	TO	TO	TO	TO	TO	MEM	MEM	MEM
Huck	10	74	301	***35.77***	296.08	3.30	4.01	3.26	***0.67***	***1.31***	11.16	2.75	MEM	**<0.1**	MEM
1dp	[23, 26]	76	769	TO	TO	TO	TO	TO	TO	TO	TO	TO	MEM	MEM	MEM
Win95	8	76	225	***32.19***	360.84	3.80	4.98	3.82	***1.36***	***1.92***	11.30	3.04	MEM	**<0.1**	MEM
Jean	10	77	254	***43.46***	320.82	3.76	4.79	4.10	***0.89***	***1.33***	12.36	2.82	MEM	**<0.1**	MEM
David	13	87	406	***103.52***	583.71	16.67	14.17	20.13	***10.08***	22.89	49.47	***17.92***	MEM	**1.88**	MEM
Cellar 02	10	100	311	***129.33***	1365.04	10.91	13.80	11.80	***1.56***	***3.26***	31.97	6.53	MEM	**<0.1**	MEM
Pathfinder	6	109	211	***105.25***	2209.20	13.36	17.37	14.10	***1.89***	***2.64***	27.22	5.26	MEM	**<0.1**	MEM
Mulsol.i.5.-pp	31	119	2556	***2539.76***	3217.04	2471.04	**514.01**	1915.01	2295.62	***2251.63***	TO	3234.98	MEM	MEM	MEM
Miles500	22	128	1170	***942.95***	4486.53	401.58	464.48	164.82	**121.72**	465.25	850.16	***264.55***	MEM	MEM	MEM