RAIRO Operations Research
RAIRO Oper. Res. **38** (2004) 13-26
DOI: 10.1051/ro:2004011

HEURISTIC AND METAHEURISTIC METHODS FOR COMPUTING GRAPH TREEWIDTH

François Clautiaux[1], Aziz Moukrim[1], Stéphane Nègre[2] and Jacques Carlier[1]

Communicated by Pierre Tolla

Abstract. The notion of treewidth is of considerable interest in relation to NP-hard problems. Indeed, several studies have shown that the tree-decomposition method can be used to solve many basic optimization problems in polynomial time when treewidth is bounded, even if, for arbitrary graphs, computing the treewidth is NP-hard. Several papers present heuristics with computational experiments. For many graphs the discrepancy between the heuristic results and the best lower bounds is still very large. The aim of this paper is to propose two new methods for computing the treewidth of graphs: a heuristic and a metaheuristic. The heuristic returns good results in a short computation time, whereas the metaheuristic (a Tabu search method) returns the best results known to have been obtained so far for all the DIMACS vertex coloring / treewidth benchmarks (a well-known collection of graphs used for both vertex coloring and treewidth problems.) Our results actually improve on the previous best results for treewidth problems in 53% of the cases. Moreover, we identify properties of the triangulation process to optimize the computing time of our method.

Keywords. Treewidth, elimination orderings, triangulated graphs, heuristic, metaheuristic, computational experiments.

Accepted December 19, 2003.

[1] Laboratoire HeuDiaSyC, UMR CNRS 6599, UTC, BP 20529, 60205 Compiègne, France; e-mail: francois.clautiaux,aziz.moukrim,jacques.carlier@hds.utc.fr

[2] Laboratoire de Recherche en Informatique d'Amiens, INSSET, 48 rue Raspail, 02100 St Quentin, France; e-mail: stephane.negre@insset.u-picardie.fr

1. Introduction

The tree-decomposition method was introduced by Robertson and Seymour as part of their research into *graph minors* [17]. The approach decomposes the representative graph $G = (V, E)$ into separating vertex subsets, called *separators*, connected in a tree. It uses the property that several states of a subgraph can be summarized by the state of a separator. For instance, in the graph coloration problem, the state of a subgraph can be summarized by a coloration of the separator [16]. In the all-terminal reliability problem, it is summarized by the decomposition of the separator into connected components [6]. The complexity of the method becomes exponential when the maximum size of a separator is not bounded by a constant.

The maximum size of a separator minus one in an optimal tree-decomposition is called *treewidth*, and is denoted $tw(G)$. The tree-decomposition method can be used to tackle NP-hard problems in large graphs with a bounded treewidth, using dynamic programming methods. It has numerous applications, including classical optimization problems [6, 15] such as graph coloration [16], probabilistic networks [11], and the frequency assignment problem [13].

Computing the treewidth of a graph is NP-hard [3]. Several exact methods deal with the decision problem $tw(G) \leq k$, but their practical interest is restricted to small values of k (see [4] for $k = 1, 2, 3$ and [5] for an arbitrary k). A reasonable way to tackle the problem is to compute good solutions using heuristic methods. The most recent papers propose approximate methods and computational experiments. It would appear that the distance between the lower and upper bounds is still large for several instances [14], and so the design of more efficient heuristics is a real challenge.

In this paper we focus on upper bounds with a greedy algorithm and a Tabu search method. We apply them using the well known DIMACS benchmark for graph coloring [12]. The results of the greedy algorithm are better on average than those obtained by the most recent methods, and require substantially less computation time. For each graph constituting the benchmark, our Tabu Search finds the best upper bound known so far. Moreover, our results are strictly better in 53% of the cases. The value returned by our method can be 25% lower than the previous best upper bound.

In Section 2 we give the notation and the definitions of the notions used in this paper. We propose a new heuristic in Section 3 and our Tabu Search Method in Section 4. The computational experiments are presented in Section 5, and finally concluding remarks and some ideas for future work in Section 6.

2. Definitions and notation

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set $E \subset V \times V$. Let $n = |V|$ and $m = |E|$. Let v be a vertex. u is a *neighbor* of v in G if $[v, u] \in E$. The set of neighbors of v is called the *neighborhood* of v,

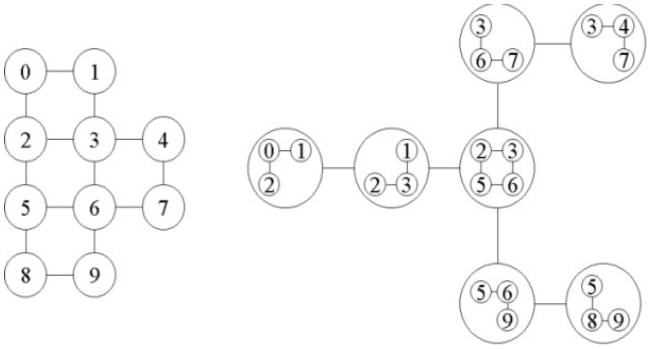

FIGURE 1. A graph and a (non optimal) tree-decomposition for this graph.

denoted $N(v)$. Let $deg(v) = |N(v)|$ be the *degree* of v. For $W \subset V$, we denote $G[W] = (W, (W \times W) \cap E)$ the *subgraph* induced by W.

A set of vertices Q is called a *clique* if there is an edge between each pair of distinct vertices of Q. The maximum cardinality of a clique in G is denoted $\omega(G)$.

For distinct vertices $u, v \in V$, a *chain* is a sequence of distinct vertices $[u = v_1, v_2, \ldots, v_j = v]$ such that for all $1 \leq i < j$, $[v_i, v_{i+1}] \in E$. A *cycle* of length j is a sequence of distinct vertices $[v = v_1, v_2, \ldots, v_j = v]$ such that for all $1 \leq i < j$, $[v_i, v_{i+1}] \in E$. A *chord* is an edge between two non-consecutive vertices in a cycle. A graph is *connected* if for each $u, v \in V$, there is a chain between u and v. A *tree* $T = (V, E)$ is a connected graph with no cycle.

Definition 2.1 (see [17])**.** A *tree-decomposition* D_T of $G = (V, E)$ is a pair (X, T), where $T = (I, F)$ is a tree with node set I and edge set F, and $X = \{X_i : i \in I\}$, is a family of subsets of V such that:

(1) $\cup_{i \in I} X_i = V$;
(2) $\forall [v, w] \in E$, there is a X_i, $i \in I$ with $v \in X_i$ and $w \in X_i$;
(3) $\forall i, j, k \in I$, if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

The *width* of a tree-decomposition is $max_{i \in I} |X_i| - 1$. The *treewidth* of a graph G, denoted by $tw(G)$, is the minimum width over all possible tree-decompositions of G.

A graph G is *triangulated* if for every cycle of length $k > 3$, there is a chord joining two non-consecutive vertices [18]. Many heuristics use the properties of triangulated graphs to find upper bounds for the treewidth, since computing the treewidth is linear in time for triangulated graphs.

Proposition 2.1 (see [8])**.** *If G is triangulated, $tw(G) = \omega(G) - 1$. Moreover, if G is triangulated, computing $\omega(G)$ and thus $tw(G)$ has complexity $O(n + m)$.*

For an arbitrary graph $G = (V, E)$, it is interesting to look for a triangulated graph which contains G. A triangulated graph $H = (V, E^T)$, with $E \subset E^T$, is called a *triangulation* for G.

Proposition 2.2 (see [19]). *For every graph G there exists a triangulation H^* such that $tw(H^*) = tw(G)$.*

Computing the treewidth of a graph G is therefore equivalent to finding such a graph H^*, and also equivalent to finding the triangulation minimizing the size of the maximum clique. This problem is NP-hard, as computing the treewidth is NP-hard, but each triangulation H of G gives an upper bound of $tw(G)$ in linear time.

Some definitions are needed to explain how a triangulation can be built. A vertex is said to be *simplicial* if its neighborhood is a clique [18]. An ordering $\sigma(1, 2, \ldots, n)$ of V (called *elimination ordering*) is a *perfect elimination ordering* for G if for any $i \in \{1, \ldots, n\}$, $\sigma(i)$ is a simplicial vertex in $G[\{\sigma(i), \ldots, \sigma(n)\}]$.

Proposition 2.3 (see [10]). *G is triangulated if and only if it has a perfect elimination ordering.*

Let $G = (V, E)$ be an arbitrary graph and σ an elimination ordering of G. We denote $H = (V, E(\sigma))$ the triangulation for G obtained by applying Algorithm 2.1 [20]. The vertices are eliminated in the elimination ordering σ. At each step i of the algorithm, the necessary edges to make $v = \sigma(i)$ be a simplicial vertex are added to the current graph. Then the vertex is deleted. Let $m' = |E(\sigma)|$. This algorithm can be implemented in $O(n + m')$ time [20]. We denote $M(v)$ the set of neighbors of v in H, and $succ(v) = \{u/u \in M(v), \sigma^{-1}(v) < \sigma^{-1}(u)\}$ the *successors* of v, *i.e.* the neighbor set of v when it is eliminated. The *predecessors* of v is the set of vertices which have v as a successor ($pred(v) = \{u/v \in succ(u)\}$).

Algorithm 2.1. Graph triangulation according to an elimination ordering σ

input: $G = (V, E)$, $\sigma(1, \ldots, n)$
output: $H = (V, E(\sigma))$
$E(\sigma) := E$
$H_0 := (V_0, E_0) = (V, E)$
for $i := 1$ to n **do**
  $E_i := E_{i-1}$
  $v := \sigma(i)$
  for u, z such that $[v, u] \in E_{i-1}$ and $[v, z] \in E_{i-1}$ **do**
    $E_i := E_i \cup \{[u, z]\}$
    $E(\sigma) := E(\sigma) \cup \{[u, z]\}$
  end for
  $V_i := V_{i-1} - \{v\}$
  $H_i := (V_i, E_i)$
end for

So, computing the treewidth can be solved the following way: find an elimination ordering which minimizes the clique size of the triangulated graph obtained by Algorithm 2.1. The treewidth is equal to $max_{v \in V} |succ(v)|$, the maximum degree of a vertex when it is eliminated. Thus, the difficulty is finding such an ordering.

3. A new heuristic for computing treewidth

Several heuristics have been proposed to compute treewidth [14]. Most of them are greedy algorithms, and the quality of the results is highly dependent on the structure of the input graph.

We propose a new heuristic which constructs an elimination ordering step by step. It is close to the well-known *minimum degree* algorithm, a classical method for triangulating a graph. Let $G = (V, E)$ be a graph. A vertex v with minimum degree is chosen. v is then eliminated after connecting its neighborhood to make a clique as in Algorithm 2.1. The process is repeated until the vertex set is empty. Locally, the choice of the vertex with minimum degree is suitable, as the size of the clique composed of its neighborhood is minimized, but the resulting overall decomposition is not optimal, globally speaking.

We propose an improvement to this method by adding a more global criterion (*cf.* Algorithm 3.1). The idea is to compute for each vertex v a lower bound for the treewidth of the graph H_i^v obtained by elimination of vertex v. We denote it $lower\_bound(H_i^v)$. We use algorithm MMD [14] to compute the lower bound because it returns good results quickly. The vertex v^* chosen is the one that minimizes the function $2 \times lower\_bound(H_i^v) + |succ(v)|$. We label this algorithm *D_LB* (see Algorithm 3.1).

Before using any heuristics the following pretreatment can be applied. The simplicial vertices are recursively deleted, and the maximum degree found is recorded. The result returned is the maximum between the maximum degree of a vertex deleted during the pretreatment and the value returned by Algorithm 3.1 applied to the remaining graph.

Algorithm 3.1. Algorithm D_LB

input: $G = (V, E)$
output: An elimination ordering σ and the width associated with this ordering.
$E(\sigma) := E$
$H_0 := (V_0, E_0) = (V, E)$
for $i := 1$ to n **do**
  $E_i := E_{i-1}$
  For each vertex $v \in V_i$, let H_i^v be the graph obtained after elimination of vertex v and $lb(H_i^v)$ a lower bound for the treewidth of graph H_i^v.
  $v^* := argmin_{v \in V}(|succ(v)| + 2 \times lb(H_i^v))$
  $\sigma(i) := v^*$
  for u, z such that $[v^*, u] \in E_{i-1}$ and $[v^*, z] \in E_{i-1}$ **do**
    $E_i := E_i \cup \{[u, z]\}$
    $E(\sigma) := E(\sigma) \cup \{[u, z]\}$

end for
$V_i := V_{i-1} - \{v^*\}$
end for

4. Metaheuristic: Tabu treewidth (TW)

Another way to find good upper bounds is to use a metaheuristic method such as a Tabu Search, which has been successfully applied to numerous combinatorial optimization problems [1].

4.1. The Tabu TW

Tabu search [9] is a neighborhood search algorithm which starts with a feasible solution and tries to improve on it iteratively. The improved solution s' is found in some neighborhood of s. The complete neighborhood is often very large. So techniques are generally used to reduce the number of neighbors actually inspected [2, 9]. To be efficient, a neighborhood search algorithm has to avoid cycling. One technique used in the Tabu search method is to keep a list L of the latest moves. If a move is in L, it is *Tabu* (*i.e.* forbidden), and it will not be tested until a fixed number of iterations has been reached.

4.1.1. *Initialization*

The initial solution is provided by algorithm MCS [21], which is a linear time algorithm. As in Section 3, the simplicial vertices can be deleted from the initial graph without modifying the treatment for the remaining graph. So the same pretreatment is applied.

4.1.2. *Solution structure*

A solution σ is an elimination ordering of V. From σ we can obtain a triangulation H for G, and thus an upper bound for $tw(G)$ in linear time (see Algorithm 2.1). Two orders σ and σ' are said to be equivalent if $H = H'$, where H (resp. H') is the graph obtained by applying the elimination process associated with σ (resp. σ').

4.1.3. *Neighborhood*

An insertion neighborhood is used. A solution σ' is a neighbor of the current solution σ if it can be obtained from σ by translating a vertex v from position i to position $j \neq i$. The size of the complete neighborhood is bounded by n^2. In the final version of our Tabu method we use a smaller neighborhood whose size is bounded by $2n$.

4.1.4. *Cost function*

The aim is to minimize the maximum cardinality of a clique in the triangulated graph H obtained by applying Algorithm 2.1. Computing $\omega(H)$ takes $O(n + m')$ time, where m' is the number of edges in H. If we use this cost function, there are too many solutions with the same cost, so we have added a second criterion. It is computed from the size of all subsets of the decomposition. Recall that $succ(v)$ is the neighbor set of v when it is eliminated (see Algorithm 2.1). The cost function is

$$f(\sigma) = n^2 \times (\omega(H))^2 + \sum_{v \in V} (|succ(v)|)^2. \quad (1)$$

Decompositions with few large subsets are thus preferred. The first criterion always dominates the second, so the function never favors a solution with a higher value of $\omega(H)$.

4.1.5. *Parameters*

The size of the Tabu list is 7. Several translations of the same vertex induce the same solution, so each move involving the last vertex inserted is Tabu, even if the new position is different from the initial one. In fact, the vertex remains Tabu during 7 iterations. The maximum number α of iterations allowed is 20000, and the algorithm stops after 10000 non-improving solutions. These values are large, and the best solution is often found after less than 500 iterations, but they are necessary for very large graphs. After having reduced the computation time of our method, the time required actually remains reasonable, even with these constants.

4.1.6. *Diversification*

If the current solution is a local optimum (*i.e.* there is no improving solution in its neighborhood) a diversification process is launched. A vertex is inserted at a random position in the ordering. As a large number of moves can be inefficient, we choose a vertex with the largest set of successors. The obtained ordering is therefore substantially different when compared to the previous ordering, but we do not resume the search from a completely different solution, which would be equivalent to restarting the method from scratch. Although our choice of a diversification is exceedingly simple, the rest of the approach is such that this choice works acceptably well.

4.2. Properties of the triangulation process

During our work on the Tabu TW, we stated properties of the triangulation process (Algorithm 2.1). We use these to optimize the size of the neighborhood and the computing time of the cost function. Indeed, in the version of TS finally selected, we only consider $O(n)$ moves for each ordering. A vertex may be exchanged with only two candidates, namely its nearest predecessor and successor in σ. The aim of this section is to justify this method.

Let

- $G = (V, E)$ be a graph, and $v, z \in V$;
- σ be an elimination ordering for V, and i be an index, $1 \leq i < n$ such that $\sigma(i) = v$ and $\sigma(i+1) = z$;
- σ' be the elimination ordering obtained from σ by exchanging the positions of the vertices v and z. So $\sigma'(i) = z$ and $\sigma'(i+1) = v$;
- H (resp. H') be the triangulated graph obtained by the triangulation process with σ (resp. σ');
- $H_i = (V_i, E_i)$ (resp. $H'_i = (V'_i, E'_i)$) be the current graph after step i of the triangulation process with the elimination ordering σ (resp. σ') (see Algorithm 2.1);
- $succ_i(v) = \{u/[v, u] \in E_i\}$ (resp. $succ'_i(v)$) be the successor set of v at step i with the elimination ordering σ (resp. σ').

We show that after step i of the triangulation algorithm the graph H_i is independent of the ordering of the vertices previously eliminated. It depends only on the set of vertices eliminated.

Recall that at each step i, if $t, u \in succ_i(\sigma(i))$, $[t, u]$ is added to the graph H_i. As the two elimination orderings σ and σ' are equal until index i, for each $j < i$ and $p \in V, succ_j(p) = succ'_j(p)$.

We now define the existence of edge $[t, u]$ in H_{i+1}.

$[t, u] \in E_{i+1}$ means either that $[t, u]$ was added at step $i+1$, or that it existed before. At least one of the following three conditions obtains, corresponding respectively to the situation where $[t, u]$ existed before step i, or was added at step i or $i+1$.

$$[t, u] \in E_{i-1} \tag{2}$$

$$t, u \in succ_{i-1}(v) \tag{3}$$

$$t, u \in succ_i(z). \tag{4}$$

Now consider the existence of $[t, u]$ at the same step with σ'. Symmetrically, its existence implies that one of these conditions obtains, corresponding respectively to the situation where $[t, u]$ existed before step i, or was added at step i or $i+1$.

$$[t, u] \in E'_{i-1} = E_{i-1} \tag{5}$$

$$t, u \in succ'_{i-1}(z) = succ_{i-1}(z) \tag{6}$$

$$t, u \in succ'_i(v). \tag{7}$$

We consider two cases, depending on the existence of $[v, z]$ in H_{i-1}.

Proposition 4.1. *Suppose we have G, σ, σ' (obtained by exchanging two consecutive vertices v and z in σ), H and H' as defined above. If v and z are not neighbors in H_{i-1} (and thus not in H), then $H = H'$.*

Proof. Suppose $[v, z] \notin E_{i-1}$. Since $z \notin succ_{i-1}(v)$, $succ'_i(v) = succ'_{i-1}(v) = succ_{i-1}(v)$ and $succ_i(z) = succ_{i-1}(z)$. Given this condition, (3) is equivalent to (7), and (4) is equivalent to (6). So $[t, u] \in E'_{i+1}$ is equivalent to $[t, u] \in E_{i+1}$.

As the elimination orderings are equal after index $i+1$, it follows that the graphs obtained with σ and σ' are identical. □

Proposition 4.2. *Let $G = (V, E)$ be a graph, σ an elimination ordering for V, and σ' the ordering obtained from σ by exchanging two consecutive vertices v and z. For all $t \notin \{v, z\}$, $H_{\sigma^{-1}(t)} = H'_{\sigma^{-1}(t)}$.*

Proof. Using Proposition 4.1, we only have to consider the situation where $[v, z] \in E_{i-1}$. (4) means that one of the following is the case, according respectively to the situation where $[z, t]$ and $[z, u]$ exist before step i, or $[z, u]$ is added at step i, $[z, t]$ added at step i, or the two edges are added at step i.

$$t, u \in succ_{i-1}(z) \tag{8}$$
$$(t \in succ_{i-1}(z)) \text{ and } (u \in succ_{i-1}(v)) \tag{9}$$
$$(u \in succ_{i-1}(z)) \text{ and } (t \in succ_{i-1}(v)) \tag{10}$$
$$t, u \in succ_{i-1}(v). \tag{11}$$

(7) means that one of the following is the case, according respectively to the situation where $[z, u]$ and $[z, t]$ exist in graph $H'_{i-1} = H_{i-1}$, or $[z, u]$ is added at step i, $[z, t]$ is added at step i, or the two edges are added at step i.

$$t, u \in succ'_{i-1}(v) = succ_{i-1}(v) \tag{12}$$
$$(t \in succ_{i-1}(v)) \text{ and } (u \in succ_{i-1}(z)) \tag{13}$$
$$(u \in succ_{i-1}(v)) \text{ and } (t \in succ_{i-1}(z)) \tag{14}$$
$$t, u \in succ_{i-1}(z). \tag{15}$$

From these cases, it can easily be verified that $[t, u] \in E'_{i+1}$ if and only if $[t, u] \in E_{i+1}$. Indeed, (2), (3), (8), (9), (10) and (11) correspond respectively to (5), (12), (15), (14), (13) and (12).

So $E'_{i+1} = E_{i+1}$. As the two sets are equal and $\sigma(j) = \sigma'(j)$ for $j > i$, no other following intermediate graph is modified. □

In the next section we explain how Proposition 4.2 can be used to reduce the size of the neighborhood actually explored. Moreover, we generalize this property using a simple inductive proof, and we apply it to the computation of the cost function.

Proposition 4.3. *Using the same notation as above, if vertex v is translated from position i to position j in σ, then we have $H_l = H'_l$ for $l \notin [min\{i, j\}, max\{i, j\}]$.*

Proof. Consider a translation of vertex v from position i to position j in σ (without loss of generality, we assume $i < j$). This translation can be broken up into $j - i$ elementary exchanges of v and the consecutive vertices $\sigma(k)$ with $k \in [i, j]$. All exchanges are in the interval $[i, j]$. So, by Proposition 4.2, the only modified graphs are the intermediate graphs H_k with $k \in [i, j]$. □

4.3. Optimizing the Tabu TW

4.3.1. *An improved neighborhood*

Let us denote respectively $\min Succ(v) = \min_{z \in succ(v)}(\sigma^{-1}(z))$ and $\max Pred(v) = \max_{z/v \in succ(z)}(\sigma^{-1}(z))$ the positions of the nearest neighbors of vertex v respectively after and before. Let σ' be the elimination ordering obtained from σ by translating vertex v from position i to position j. If $\max Pred(v) < j < \min Succ(v)$, σ' is equivalent to σ (Prop. 4.1). The translation of v at position $\max Pred(v)$ is therefore the first significant move to a lower position in the elimination ordering. This observation also applies to $\min Succ(v)$. This property can be recursively applied to reduce the size of the neighborhood. We have described a large number of classes of solutions which are strictly equivalent. Thus, a new neighborhood is: for each vertex, the only necessary translations to test are those at the position of its neighbors in the current graph H obtained with σ.

When a translation is tested at position $\min Succ(v)$, v could have new neighbors added by the elimination of $\sigma(\min Succ(v))$. Consequently we have to determine the new set of neighbors for each translation after $\min Succ(v)$. Another solution is simply to ignore these new neighbors in order to shorten the computation time. This produces a neighborhood whose size is bounded by m' (one move for each edge), but some information is lost. It makes the method scalable. Note that a translation at $\max Pred(v)$ cannot increase the number of neighbors of v.

4.3.2. *An $O(n)$ Neighborhood*

In the previous paragraph we defined *the smallest significant move* as the translation of v at the position of its nearest neighbors (minSucc and maxPred). Every move can be broken down into several *smallest significant moves*. So we can design a new neighborhood: for each vertex v, the translations tested are those at $\min Succ(v)$ and $\max Pred(v)$. The size of the neighborhood is bounded by $2n$. This last method has a time complexity bounded by $\alpha n(m' + n)$, where prefixed parameter α is the maximum number of iterations allowed. It finds upper bounds for the treewidths of large graphs, and our computing times are not significantly different from those of the well known Lex greedy heuristics, and from the MSVS heuristic [14] which uses network flow techniques to find minimal separating vertex sets.

4.3.3. *Application for the computation of the cost function*

Let σ be an elimination ordering of V, and σ' the ordering obtained from σ by the translation of vertex v from position i to position j. Let $succ(v)$ (resp. $succ'(v)$) be the successors of v with σ (resp. σ'). We have, from Proposition 4.3, that for each z such that $\sigma^{-1}(z) \notin [\min\{i,j\}, \max\{i,j\}]$, $succ(z) = succ'(z)$. Thus, by recording the previous values of $|succ(z)|$ for each z in V, the only ones to compute again for σ' are those related to the vertices in $[\min\{i,j\}, \max\{i,j\}]$.

As a quadratic sum is involved (*cf.* Eq. (1)), the values of $|succ(v)|$ are not sufficient for us to be able to avoid taking all vertices into account, because we

still have to sum up the values of $|succ(v)|^2$. To avoid computing the quadratic sum entirely, the following intermediate sum $\sum_{l=k}^{n} |succ(\sigma(l))|^2$ is recorded for each position k in the ordering. A similar technique is used to compute $\omega(H)$. The value $\max_{k \leq l \leq n} |succ(\sigma(l))|$ is recorded for each position k. Thus the triangulation process can stop at index $\max\{i, j\}$. The total sum can be obtained by summing $\sum_{l=1}^{k} |succ'(\sigma(l))|^2$, the intermediate sum obtained and $\sum_{l=k+1}^{n} |succ(\sigma(l))|^2$ the recorded value. We could begin at position $\min\{i, j\}$, but we would have to record a large number of intermediate graphs, which would not be any faster.

4.3.4. *List of non improving neighbors*

The idea is to avoid testing moves that have been tested in a previous iteration. Let σ be a solution and H its associated triangulated graph. With the $O(n)$ neighborhood, we only test two translations for each vertex. Now let σ' (associated to H' and σ') be the solution obtained after translating vertex v from position i to position j in σ. From Proposition 4.3, if a move m of a vertex from position k to position l with $k, l \notin [\min\{i, j\}, \max\{i, j\}]$ does not improve solution σ, it does not improve solution σ' either. The intermediate graphs modified by the move are in fact the same.

So, for each vertex v, we record whether or not the translation to position $\max Pred(v)$ or $\min Succ(v)$ improves the current solution. The moves involving positions k and l which do not improve the previous solution are not tested if $\max\{k, l\} < \min\{i, j\}$ or $\min\{k, l\} > \max\{i, j\}$.

5. Computational analysis

We have tested our methods on the DIMACS benchmarks for vertex coloring. So, we can compare them to the most recent heuristics (Lex and MSVS) [14]. The lower bound is provided by algorithm *LB_P* [7] which returns the best lower bounds known for this benchmark. Our algorithm is implemented in C on a Pentium III 1GHz, whereas the computation time of Lex and MSVS were obtained with a C++ implementation on a Pentium III 800 Mhz.

The results of the heuristic are on average as good as those returned by MSVS and Lex, and our computing times are far smaller for the large instances, even taking into account the different speeds of the two computers. It finds good upper bounds in a short computation time.

As far as we know, the results returned by the *Tabu TW* method are the best upper bounds achieved for the DIMACS benchmarks. Note that the values returned by the Tabu search are never larger than the previous bests. Moreover, a better upper bound is found in 53% of the cases, with improvements as high as 25%. The computation time of this method is longer than for the other methods, in view of the high value of the constant α. The best value is often found after several iterations, and although this constant may be reduced for small graphs, results may be less impressive. Our computational experiments are reported in Table 1.

TABLE 1. Computational experiments for the DIMACS Benchmark.

instance	n	m	*lb*	ub				CPU time			
			LB_P	LEX	MSVS	D_LB	Tabu	LEX	MSVS	D_LB	Tabu
Anna	138	986	*11*	12	12	12	12	1.24	18.39	0.880	2776.93
David	87	812	*11*	13	13	13	13	0.56	7.77	0.220	796.81
Huck	74	602	*10*	10	10	10	10	0.24	2.30	0.130	488.76
Homer	561	3258	*21*	37	31	32	31	68.08	556.82	27.380	157716.56
Jean	80	508	*9*	9	9	9	9	0.29	1.98	0.130	513.76
Games120	120	638	*12*	37	51	41	33	5.20	65.97	1.620	2372.71
QUEEN5_5	25	160	*12*	18	18	18	18	0.04	0.22	0.340	100.36
QUEEN6_6	36	290	*15*	26	28	27	25	0.16	1.16	0.140	225.55
QUEEN7_7	49	476	*20*	35	38	38	35	0.51	4.66	0.090	322.40
QUEEN8_8	64	728	*23*	46	49	50	46	1.49	16.38	0.350	617.57
QUEEN9_9	81	1056	*26*	59	66	64	58	3.91	47.35	0.740	1527.13
QUEEN10_10	100	1470	*31*	73	79	80	72	9.97	128.30	1.670	3532.78
QUEEN11_11	121	1980	*34*	89	101	102	88	23.36	310.83	3.160	5395.74
QUEEN12_12	144	2596	*37*	106	120	117	104	49.93	702.29	6.720	10345.14
QUEEN13_13	169	3328	*42*	125	145	141	122	107.62	1589.77	10.940	16769.58
QUEEN14_14	196	4186	*45*	145	164	164	141	215.36	3275.75	20.300	29479.91
QUEEN15_15	225	5180	*48*	167	192	194	163	416.25	6002.33	31.070	47856.25
QUEEN16_16	256	6320	*53*	191	214	212	186	773.09	11783.30	63.890	73373.12
FPSOL2.I.1	269	11654	*66*	66	66	66	66	319.34	4220.91	176.110	63050.58
FPSOL2.I.2	363	8691	*31*	52	31	31	31	622.22	8068.88	174.930	78770.05
FPSOL2.I.3	363	8688	*31*	52	31	31	31	321.89	8131.78	144.650	79132.70
INITHX.I.1	519	18707	*56*	223	56	56	56	3144.95	37455.10	2966.020	101007.52
INITHX.I.2	558	13979	*31*	228	35	35	35	5567.96	37437.20	1004.340	121353.69
INITHX.I.3	559	13969	*31*	228	35	35	35	5190.39	36566.80	884.430	119080.85
MILES1000	128	3216	*48*	49	53	53	49	14.39	229.00	3.420	5696.73
MILES1500	128	5198	*76*	77	83	77	77	29.12	268.19	3.470	6290.44
MILES250	125	387	*8*	10	9	9	9	1.12	10.62	0.350	1898.29
MILES500	128	1170	*22*	22	28	28	22	4.37	87.18	0.960	4659.31
MILES750	128	2113	*33*	37	38	43	36	8.13	136.69	1.850	3585.68
MULSOL.I.1	138	3925	*50*	66	50	50	50	17.77	240.24	12.700	3226.77
MULSOL.I.2	173	3885	*32*	69	32	32	32	34.06	508.71	15.290	12310.37
MULSOL.I.3	174	3916	*32*	69	32	32	32	34.58	527.89	14.010	9201.45
MULSOL.I.4	175	3946	*32*	69	32	32	32	35.53	535.72	14.100	8040.28
MULSOL.I.5	176	3973	*31*	69	31	31	31	36.25	549.55	12.920	13014.81
MYCIEL3	11	20	*4*	5	5	5	5	0.00	0.01	0.000	72.50
MYCIEL4	23	71	*6*	11	11	11	10	0.02	0.13	0.310	84.31
MYCIEL5	47	236	*12*	23	20	20	19	0.28	2.00	0.300	211.73
MYCIEL6	95	755	*20*	47	35	35	35	4.56	29.83	2.410	1992.42
MYCIEL7	191	2360	*31*	94	74	70	66	109.86	634.32	28.640	19924.58
SCHOOL1	385	19095	*116*	252	244	242	188	3987.64	41141.10	273.620	137966.73
SCHOOL1_NSH	352	14612	*100*	192	214	200	162	2059.52	28954.90	161.700	180300.10
ZEROIN.I.1	126	4100	*50*	50	50	50	50	17.78	338.26	10.680	2595.92
ZEROIN.I.2	157	3541	*31*	40	33	33	32	24.82	448.74	26.760	4825.51
ZEROIN.I.3	157	3540	*31*	40	33	33	32	24.69	437.06	24.780	8898.80
LE450_5A	450	5714	*33*	310	317	323	256	7836.99	73239.66	274.490	130096.77
LE450_5B	450	5734	*33*	313	320	321	254	7909.11	73644.28	260.290	187405.33
LE450_5C	450	9803	*51*	348	340	329	272	10745.70	103637.17	525.350	182102.37
LE450_5D	450	9757	*51*	349	326	318	278	10681.29	96227.40	566.610	182275.69
LE450_15A	450	8168	*56*	296	297	300	272	6887.15	59277.90	273.700	117042.59
LE450_15B	450	8169	*55*	296	307	305	270	6886.84	65173.20	230.900	197527.14
LE450_15C	450	16680	*92*	379	376	379	359	12471.09	122069.00	356.610	143451.73
LE450_15D	450	16750	*91*	379	375	380	360	12481.22	127602.00	410.350	117990.30
LE450_25A	450	8260	*62*	255	270	267	234	4478.30	53076.40	243.290	143963.41
LE450_25B	450	8263	*59*	251	264	266	233	4869.97	52890.00	248.610	184165.21
LE450_25C	450	17343	*100*	355	365	361	327	10998.68	109141.00	344.360	151719.58
LE450_25D	450	17425	*98*	356	359	362	336	11376.02	111432.25	434.120	189175.40
DSJC125.1	125	736	*16*	70	67	65	65	12.90	171.54	2.500	1532.93
DSJC125.5	125	3891	*62*	110	110	110	109	38.07	254.90	3.870	2509.97
DSJC125.9	125	6961	*108*	119	120	120	119	55.60	70.79	56.630	1623.44
DSJC250.1	250	3218	*32*	183	179	176	173	528.10	5507.86	32.730	28606.12
DSJC250.5	250	15668	*125*	233	233	233	232	1111.66	7756.38	48.510	14743.35
DSJC250.9	250	27897	*218*	243	244	244	243	1414.58	1684.83	15.600	30167.70

6. Conclusion

We propose two methods for computing upper bounds for graph treewidths. The heuristic gives good results in a short computation time, whereas the Tabu search returns the best upper bounds known so far for the DIMACS benchmarks. Our results, even in the worst case, are at least as good as the previous best results. In addition, we note that the Tabu search includes strategies that we have not examined, and taking these into account may improve our results yet further.

The gap between our upper bounds and the best lower bounds is still large. The exact value of treewidth is still unknown for most of the benchmarks. In order to reduce the distance between the bounds we need first to raise the value of the lower bound, before we set about developing an exact method.

In the future we also plan to adapt the Tabu TW for computing the pathwidth of graphs. We need to modify the cost function and to identify new properties involving interval graphs instead of triangulated graphs.

Acknowledgement. The authors would like to thank the referees for their constructive comments.

References

[1] E. Aarts and J.K. Lenstra, Local Search in Combinatorial Optimization. *Series in Discrete Mathematics and Optimization*. John Wiley and Sons (1997).
[2] R. Ahuja, O. Ergun, J. Orlin and A. Punnen, A survey on very large-scale neighborhood search techniques. *Discrete Appl. Math.* **123** (2002) 75-102.
[3] S. Arnborg, D.G. Corneil and A. Proskurowski, Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.* **8** (1987) 277-284.
[4] S. Arnborg and A. Proskurowki, Characterisation and recognition of partial 3-trees. *SIAM J. Alg. Disc. Meth.* **7** (1986) 305-314.
[5] H. Bodlaender, A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.* **25** (1996) 1305-1317.
[6] J. Carlier and C. Lucet, A decomposition algorithm for network reliability evaluation. *Discrete Appl. Math.* **65** (1993) 141-156.
[7] F. Clautiaux, J. Carlier, A. Moukrim and S. Nègre, New lower and upper bounds for graph treewidth. WEA 2003, *Lect. Notes Comput. Sci.* **2647** (2003) 70-80.
[8] F. Gavril, Algorithms for minimum coloring, maximum clique, minimum coloring cliques and maximum independent set of a chordal graph. *SIAM J. Comput.* **1** (1972) 180-187.
[9] F. Glover and M. Laguna, *Tabu search*. Kluwer Academic Publishers (1998).
[10] M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York (1980).
[11] F. Jensen, S. Lauritzen and K. Olesen, Bayesian updating in causal probabilistic networks by local computations. *Comput. Statist. Quaterly* **4** (1990) 269-282.
[12] D.S. Johnson and M.A. Trick, The Second DIMACS Implementation Challenge: NP-Hard Problems: Maximum Clique, Graph Coloring, and Satisfiability. *Series in Discrete Math. Theor. Comput. Sci. Amer. Math. Soc.* (1993).
[13] A. Koster, *Frequency Assignment, Models and Algorithms*. Ph.D. Thesis, Universiteit Maastricht (1999).
[14] A. Koster, H. Bodlaender and S. van Hoesel, Treewidth: Computational experiments. *Fund. Inform.* **49** (2001) 301-312.

[15] C. Lucet, J.F. Manouvrier and J. Carlier, Evaluating network reliability and 2-edge-connected reliability in linear time for bounded pathwidth graphs. *Algorithmica* **27** (2000) 316-336.
[16] C. Lucet, F. Mendes and A. Moukrim, Méthode de décomposition appliquée à la coloration de graphes, in *ROADEF* (2002).
[17] N. Robertson and P. Seymour, Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms* **7** (1986) 309-322.
[18] D. Rose, Triangulated graphs and the elimination process. *J. Math. Anal. Appl.* **32** (1970) 597-609.
[19] D. Rose, A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations, in *Graph Theory and Computing*, edited by R.C. Reed. Academic Press (1972) 183-217.
[20] D. Rose, R. Tarjan and G. Lueker, Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.* **5** (1976) 146-160.
[21] R. Tarjan and M. Yannakakis, Simple linear-time algorithm to test chordality of graphs, test acyclicity of hypergraphs, and selectivity reduce acyclic hypergraphs. *SIAM J. Comput.* **13** (1984) 566-579.