Practical Algorithms for Linear Boolean-width*

Chiel B. ten Brinke, Frank J. P. van Houten, and Hans L. Bodlaender

Department of Computer Science, Utrecht University
PO Box 80.089, 3508 TB Utrecht, The Netherlands
CtenBrinke@gmail.com, Frankv@nhouten.com, H.L.Bodlaender@uu.nl

Abstract

In this paper, we give a number of new exact algorithms and heuristics to compute linear boolean decompositions, and experimentally evaluate these algorithms. The experimental evaluation shows that significant improvements can be made with respect to running time without increasing the width of the generated decompositions. We also evaluated dynamic programming algorithms on linear boolean decompositions for several vertex subset problems. This evaluation shows that such algorithms are often much faster (up to several orders of magnitude) compared to theoretical worst case bounds.

1998 ACM Subject Classification G.2.2 [Discrete Mathematics] Graph Theory – Graph algorithms, F.2.2 [Analysis of Algorithms and Problem Complexity] Nonnumerical Algorithms and Problems – Computations on discrete structures

Keywords and phrases graph decomposition, boolean-width, heuristics, exact algorithms, vertex subset problems

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.187

1 Introduction

Boolean-width is a recently introduced graph parameter [2]. Similarly to treewidth and other parameters, it measures some structural complexity of a graph. Many NP-hard problems on graphs become easy if some graph parameter is small. We need a derived structure which captures the necessary information of a graph in order to exploit such a small parameter. In the case of boolean-width, this is a binary partition tree, referred to as the decomposition tree. However, computing an optimal decomposition tree is usually a hard problem in itself. A common approach to bypass this problem is to use heuristics to compute decompositions with a low boolean-width.

Algorithms for computing boolean decompositions have been studied before in [16, 8, 10, 5], but in this paper we study the specific case of linear boolean decompositions, which are considered in [1, 8, 10]. Linear decompositions are easier to compute and the theoretical running time of algorithms for solving practical problems is lower on linear decompositions than on tree shaped ones. For instance, vertex subset problems can be solved in $O^*(nec^3)$ due to a dynamic programming algorithm by Bui-Xuan et al. [3], but this can be improved to $O^*(nec^2)$ for linear decompositions. Here, nec is the number of d-neighborhood equivalence classes, i.e., the maximum size of the dynamic programming table.

* The research of the third author was partially funded by the Networks programme, funded by the Dutch Ministry of Education, Culture and Science through the Netherlands Organisation for Scientific Research.

10th International Symposium on Parameterized and Exact Computation (IPEC 2015).
Editors: Thore Husfeldt and Iyad Kanj; pp. 187–198

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We first give an exact algorithm for computing optimal linear boolean decompositions, improving upon existing algorithms, and subsequently investigate several new heuristics through experiments, improving upon the work by Sharmin [10, Chapter 8]. We then study the practical relevance of these algorithms experimentally by solving an instance of a vertex subset problem, investigating the number of equivalence classes compared to the theoretical worst case bounds. Omitted proofs can be found in the full version of this paper [12].

2 Preliminaries

A *graph* $G = (V, E)$ of size n is a pair consisting of a set of n *vertices* V and a set of *edges* E. The *neighborhood* of a vertex $v \in V$ is denoted by $N(v)$. For a subset $A \subseteq V$ we denote the neighborhood by $N(A) = \bigcup_{v \in A} N(v)$. In this paper we only consider simple, undirected graphs and assume we are given a total ordering on the vertices of a graph G. For a subset $A \subseteq V$ we denote the *complement* by $\overline{A} = V \setminus A$. A partition $(A, \overline{A})$ of V is called a *cut* of the graph. Each cut $(A, \overline{A})$ of G induces a bipartite subgraph $G[A, \overline{A}]$. The *neighborhood across a cut* $(A, \overline{A})$ for a subset $X \subseteq A$ is defined as $N(X) \cap \overline{A}$.

▶ **Definition 1** (Unions of neighborhoods)**.** Let $G = (V, E)$ be a graph and $A \subseteq V$. We define the set of *unions of neighborhoods across a cut* $(A, \overline{A})$ as

$$\mathcal{UN}(A) = \left\{ N(X) \cap \overline{A} \,\middle|\, X \subseteq A \right\}.$$

The number of unions of neighborhoods is symmetric for a cut $(A, \overline{A})$, i.e., $|\mathcal{UN}(A)| = |\mathcal{UN}(\overline{A})|$ [6, Theorem 1.2.3]. Furthermore, for any cut $(A, \overline{A})$ of a graph G it holds that $|\mathcal{UN}(A)| = \#\mathcal{MIS}(G[A, \overline{A}])$, where $\#\mathcal{MIS}(G)$ is the number of maximal independent sets in G [16, Theorem 3.5.5].

▶ **Definition 2** (Decomposition tree)**.** A *decomposition tree* of a graph $G = (V, E)$ is a pair (T, δ), where T is a full binary tree and δ is a bijection between the leaves of T and vertices of V. If a is a node and L are its leaves, we write $\delta(a) = \bigcup_{l \in L} \delta(l)$. So, for the root node r of T it holds that $\delta(r) = V$. Furthermore, if nodes a and b are children of a node w, then $(\delta(a), \delta(b))$ is a partition of $\delta(w)$.

In this paper we consider a special type of decompositions, namely *linear decompositions*.

▶ **Definition 3** (Linear decomposition)**.** A *linear decomposition*, or *caterpillar decomposition*, is a decomposition tree (T, δ) where T is a full binary tree and for which each internal node of T has at least one leaf as a child. We can define such a linear decomposition through a linear ordering $\pi = \pi_1, \ldots, \pi_n$ of the vertices of G by letting δ map the i-th leaf of T to π_i.

▶ **Definition 4** (Boolean-width)**.** Let $G = (V, E)$ be a graph and $A \subseteq V$. The *boolean dimension of* A is a function bool-dim : $2^V \to \mathbb{R}$.

$$\text{bool-dim}(A) = \log_2 |\mathcal{UN}(A)|.$$

Let (T, δ) be a decomposition of a graph G. We define the *boolean-width* of (T, δ) as the maximum boolean dimension over all cuts induced by nodes of (T, δ).

$$\text{boolw}(T, \delta) = \max_{w \in T} \text{bool-dim}(\delta(w))$$

The boolean-width of G is defined as the minimum boolean-width over all possible full decompositions of G, while the *linear boolean-width* of a graph $G = (V(G), E(G))$ of size n is defined as the the minimum boolean-width over all linear decompositions of G.

$$\text{boolw}(G) = \min_{(T,\delta)\ of\ G} \text{boolw}(T, \delta)$$

$$\text{lboolw}(G) = \min_{\text{linear } (T,\delta) \text{ of } G} \text{boolw}(T, \delta)$$

It is known that for any graph G it holds that $\text{boolw}(G) \leq \textit{treewidth}(G)+1$ [16, Theorem 4.2.8]. The linear variant of treewidth is called *pathwidth* [9], or pw for short.

▶ **Theorem 5.** *For any graph G it holds that* $\text{lboolw}(G) \leq \text{pw}(G) + 1$.

The algorithms in this paper make extensive use of sets and set operations, which can be implemented efficiently by using bitsets. We assume that bitset operations take $O(n)$ time and need $O(n)$ space, even though in practice this may come closer to $O(1)$. If one assumes that these requirements are constant, several time and space bounds in this paper improve by a factor n.

In this paper we assume that the graph G is connected, since if the graph consists of multiple connected components we can simply compute a linear decomposition for each connected component, after which we glue them together, in any arbitrary order.

3 Exact Algorithms

We can characterize the problem of finding an optimal linear decomposition by the following recurrence relation, in which P is a function mapping a subset of vertices A to the linear boolean-width of the induced subgraph $G[A, \overline{A}]$.

$$\begin{aligned} P(\{v\}) = |\mathcal{UN}(\{v\})| &= \begin{cases} 1 & \text{if } N(v) = \emptyset \\ 2 & \text{if } N(v) \neq \emptyset \end{cases} \\ P(A) &= \min_{v \in A}\{\max\{|\mathcal{UN}(A)|, P(A \setminus \{v\})\}\} \end{aligned} \tag{1}$$

The boolean-width of the graph G is now given by $\log_2(P(V))$. Adaptation of existing techniques lead to the following algorithms for linear boolean-width, upon we hereafter improve:

- With dynamic programming a running time of $O(2.7284^n)$ is achieved [12, Theorem 19].
- With adaptation of the exact algorithm for boolean-width by Vatshelle [16], a running time of $O(n^3 \cdot 2^{n+\text{lboolw}(G)})$ is achieved [12, Theorem 20].

3.1 Improving the running time

We present a faster and easier way to precompute for all cuts $A \subseteq V$ the value $|\mathcal{UN}(A)|$, which results in a new algorithm displayed in Algorithm 2. In the following it is important that the $\mathcal{UN}$ sets are implemented as hashmaps, which will only save distinct neighborhoods.

Algorithm 1 Compute $\mathcal{UN}(X \cup \{v\})$ given $\mathcal{UN}(X)$.

```
1: procedure INCREMENT-UN(G, X, UN_X, v)
2:     U ← ∅
3:     for S ∈ UN_X do
4:         U ← U ∪ {S \ {v}}
5:         U ← U ∪ {(S \ {v}) ∪ (N(v) ∩ (X̄ \ {v}))}
6:     return U
```

▶ **Lemma 6.** *The procedure Increment-UN is correct and runs in $O(n \cdot |\mathcal{UN}_X|)$ time using $O(n \cdot |\mathcal{UN}_X|)$ space.*

Algorithm 2 Return $\text{lboolw}(G)$, if it is smaller than $\log K$, otherwise return ∞.

```
1:  procedure Incremental-UN-exact(G, K)
2:      T_UN(∅) ← 0
3:      Compute-count-UN(G, K, T_UN, ∅, {∅})
4:
5:      P(X) ← ∞, for all X ⊆ V
6:      P(∅) ← 0
7:
8:      for i ← 0, ..., |V| − 1 do
9:          for X ⊆ V of size i do
10:             for v ∈ V \ X do
11:                 Y ← X ∪ {v}
12:                 if P(X) ≤ K then
13:                     P(Y) ← min(P(Y), max(T_UN(Y), P(X)))
14:
15:     return log2(P(V))
16:
17: procedure Compute-count-UN(G, K, T_UN, X, UN_X)
18:     for v ∈ V \ X do
19:         Y ← X ∪ {v}
20:         if T_UN(Y) is not defined then
21:             UN_Y ← Increment-UN(G, X, UN_X, v)
22:             T_UN(Y) ← |UN_Y|
23:             if T_UN(Y) ≤ K then
24:                 Compute-count-UN(G, K, T_UN, Y, UN_Y)
```

▶ **Theorem 7.** *Given a graph G, Algorithm 2 can be used to compute* $\text{lboolw}(G)$ *in* $O(n \cdot 2^{n+\text{lboolw}(G)})$ *time using* $O(n \cdot 2^n)$ *space.*

This new algorithm improves upon the previously best time [12, Theorem 20] by a factor n^2, while the space requirements stay the same. Since the tightest known upperbound for linear boolean-width is $\frac{n}{2} - \frac{n}{143} + O(1)$ [8], this algorithm can be slower than dynamic programming, since $O(2^{n+\frac{n}{2}-\frac{n}{143}+O(1)}) = O(2.8148^{n+O(1)}) \supsetneq O(2.7284^n)$, but this is very unlikely to happen in practice.

4 Heuristics

4.1 Generic form of the heuristics

The goal when using a heuristic is to find a linear ordering of the vertices in a graph in such a way that the decomposition that corresponds to this ordering will be of low boolean-width. A basic strategy to accomplish this is to start the ordering with some vertex and then by some selection criteria append a new vertex to the ordering that has not been appended yet. This strategy is used in heuristics introduced by Sharmin [10, Chapter 8].

At any point in the algorithm we denote the set of all vertices contained in the ordering by *Left*, and the remaining vertices by *Right*. While *Right* is not empty, we choose a vertex from a candidate set *Candidates* $\subseteq$ *Right*, based on a set of trivial cases, and, if no trivial case applies, by making a local greedy choice using a score function that indicates the quality of the current state *Left*, *Right*.

The choice of the initial vertex can be of great influence on the quality of the decomposition. Sharmin proposes to use a double breadth first search (BFS) in order to select this vertex, but since we will see in Chapter 5 that applications are a lot more expensive in terms of running time, it is wise to use all possible starting vertices when trying to find a good decomposition.

4.1.1 Pruning

Starting from multiple initial vertices allows us to do some pruning. If we notice during the algorithm that the score of the decomposition that is being constructed exceeds the score of the best decomposition found so far, we can stop immediately and move to the next initial vertex. For this reason, it is wise to start with the most promising initial vertices (e.g. obtained by the double BFS method), and after that try all other initial vertices.

4.1.2 Candidates

The most straightforward choice for the set *Candidates* is to take *Right* entirely. However, we may do unnecessary work here, since vertices that are more than 2 steps away from any vertex in $Left$ cannot decrease the size of $\mathcal{UN}$. This means that they should never be chosen by a greedy score function, which means that we can skip them right away. By this reasoning, the set of *Candidates* can be reduced to $N^2(Left) \cap Right = N(Left \cup N(Left)) \cap Right$. Especially for larger sparse graphs, this can significantly decrease the running time.

4.1.3 Trivial cases

A vertex is chosen to be the next vertex in the ordering if it can be guaranteed that it is an optimal choice by means of a trivial case. Lemma 8 generalizes results by Sharmin [10], since the two trivial cases given by her are subcases of our lemma, namely $X = \emptyset$ and $X = \{u\}$ for all $u \in Left$. Note that we can add a wide range of trivial cases by varying X, such as $X = Left$ and $\forall u, w \in Left : X = \{u, w\}$, but this will increase the complexity of the algorithm.

▶ **Lemma 8.** *Let $X \subseteq Left$. If $\exists v \in Right$ such that $N(v) \cap Right = N(X) \cap Right$, then choosing v will not change the boolean-width of the resulting decomposition.*

4.1.4 Relative Neighborhood Heuristic

For a cut $(Left, Right)$ and a vertex v define

$$Internal(v) = (N(v) \cap N(Left)) \cap Right$$
$$External(v) = (N(v) \setminus N(Left)) \cap Right$$

In the original formulation by Sharmin [10] $\frac{|External(v)|}{|Internal(v)|}$ is used as a score function. However, if we use $\frac{|External(v)|}{|Internal(v)|+|External(v)|} = \frac{|External(v)|}{|N(v) \cap Right|}$ we get the same ordering by Lemma 9, without having an edge case for dividing by zero. Furthermore, in contrast to Sharmin's proposal of checking for each vertex $w \in N(v)$ if $w \in N(Left) \cap Right$ or not, we can compute these sets directly by performing set operations. We will refer to this heuristic by RelativeNeighborhood.

▶ **Lemma 9.** *The mapping $\frac{a}{b} \mapsto \frac{a}{a+b}$ is order preserving.*

Two variations on this heuristic can be obtained through the score functions $\frac{|External(v)|}{|N(v)|}$ and $1 - \frac{|Internal(v)|}{|N(v)|}$, which work slightly better for sparse random graphs and extremely well for dense random graphs respectively. We will refer to these two variations by RelativeNeighborhood$_2$ and RelativeNeighborhood$_3$.

One can easily see that the running time of these three algorithms is $O(n^3)$ and the required space amounts to $O(n)$. Notice however that this algorithm only gives us a decomposition. If we need to know the corresponding boolean-width we need to compute it afterwards, for instance by iteratively applying Increment-UN on the vertices in the decomposition, and taking the maximum value. This would require an additional $O(n^2 \cdot 2^k)$ time and $O(n \cdot 2^k)$ space, where k is the boolean-width of the decomposition.

4.1.5 Least Cut Value Heuristic

The LeastCutValue heuristic by Sharmin [10] greedily selects the next vertex $v \in Right$ that will have the smallest boolean dimension across the cut $(Left \cup \{v\}, Right \setminus \{v\})$. This vertex is obtained by constructing the bipartite graph $BG = G[Left \cup \{v\}, Right \setminus \{v\}]$ for each $v \in Right$, and counting the number of maximal independent sets of BG using the CCM_{IS} [7] algorithm on BG, with the time of CCM_{IS} being exponential in n.

4.1.6 Incremental Unions of Neighborhoods Heuristic

Generating a bipartite graph and then calculating the number of maximal independent sets is a computational expensive approach. A different way to compute the boolean dimension of each cut is by reusing the neighborhoods from the previous cut, similarly to Incremental-UN-exact. We present a new algorithm, called the Incremental-UN-heuristic, in Algorithm 3. A useful property of this algorithm is that the running time is output sensitive. It follows that if a decomposition is not found within reasonable time, then the decomposition that would have been generated is not useful for practical algorithms.

▶ **Theorem 10.** *The* Incremental-UN-heuristic *procedure runs in $O(n^3 \cdot 2^k)$ time using $O(n \cdot 2^k)$ space, where k is the boolean-width of the resulting linear decomposition.*

5 Vertex subset problems

Boolean decompositions can be used to efficiently solve a class of vertex subset problems called (σ, ρ) vertex subset problems, which were introduced by Telle [11]. This class of problems consists of finding a (σ, ρ)-set of maximum or minimum cardinality and contains well known problems such as the maximum independent set, the minimum dominating set and the maximum induced matching problem. The running time of the algorithm for solving these problems is $O(n^4 \cdot nec_d(T, \delta)^3)$ [3], where $nec_d(T, \delta)$ is the number of equivalence classes of a problem specific equivalence relation, which can be bounded in terms of boolean-width. In Section 6 we investigate how close the value of $nec_d(T, \delta)$ comes to any of the theoretical bounds.

Algorithm 3 Greedy heuristic that incrementally keeps track of the Unions of Neighborhoods.

```
1:  procedure INCREMENTAL-UN-HEURISTIC(G, init)
2:      Decomposition ← (init)
3:      Left, Right ← {init}, V \ {init}
4:      UN_Left ← {∅, N(init) ∩ Right}
5:      while Right ≠ ∅ do
6:          Candidates ← set returned by candidate set strategy
7:          if there exists v ∈ Candidates belonging to a trivial case then
8:              chosen ← v
9:              UN_chosen ← INCREMENT-UN(G, Left, UN_Left, v)
10:         else
11:             for all v ∈ Candidates do
12:                 UN_v ← INCREMENT-UN(G, Left, UN_Left, v)
13:                 if chosen is undefined or |UN_v| < |UN_chosen| then
14:                     chosen ← v
15:                     UN_chosen ← UN_v
16:         Decomposition ← Decomposition · chosen
17:         Left ← Left ∪ {chosen}
18:         Right ← Right \ {chosen}
19:         UN_Left ← UN_chosen
20:     return Decomposition
```

5.1 Definitions

▶ **Definition 11** ((σ,ρ)-set)**.** Let $G=(V,E)$ be a graph. Let σ and ρ be finite or co-finite subsets of $\mathbb{N}$. A subset $X \subseteq V$ is called a (σ,ρ)-set if the following holds

$$\forall v \in V : |N(v) \cap X| \in \begin{cases} \sigma & \text{if } v \in X, \\ \rho & \text{if } v \in V \setminus X. \end{cases}$$

In order to confirm if a set X is a (σ,ρ)-set we have to count the number of neighbors each vertex $v \in V$ has in X, where it suffices to count up until a certain number of neighbors. As an example, when we want to confirm if a set X is an independent set, which is equivalent to checking if X is a $(\{0\},\mathbb{N})$-set, it is irrelevant if a vertex v has more than one neighbor in X. We capture this property in the function $d : 2^{\mathbb{N}} \to \mathbb{N}$, which is defined as follows:

▶ **Definition 12** (d-function)**.** Let $d(\mathbb{N}) = 0$. For every finite or co-finite set $\mu \subseteq \mathbb{N}$, let $d(\mu) = 1 + \min(\max_{x\in\mathbb{N}} x : x \in \mu, \max_{x\in\mathbb{N}} x : x \notin \mu)$. Let $d(\sigma,\rho) = \max(d(\sigma), d(\rho))$.

▶ **Definition 13** (d-neighborhood)**.** Let $G=(V,E)$ be a graph. Let $A \subseteq V$ and $X \subseteq A$. The *d-neighborhood* of X with respect to A, denoted by $N_A^d(X)$, is a multiset of vertices from $\overline{A}$, where a vertex $v \in \overline{A}$ occurs $\min(d, |N(v) \cap X|)$ times in $N_A^d(X)$. A d-neighborhood can be represented as a vector of length $|\overline{A}|$ over $\{0,1,\ldots,d\}$.

▶ **Definition 14** (d-neighborhood equivalence)**.** Let $G=(V,E)$ be a graph and $A \subseteq V$. Two subsets $X, Y \subseteq A$ are said to be *d-neighborhood equivalent* with respect to A, denoted by $X \equiv_A^d Y$, if it holds that $\forall v \in \overline{A} : \min(d, |X \cap N(v)|) = \min(d, |Y \cap N(v)|)$. The number of equivalence classes of a cut $(A,\overline{A})$ is denoted by $nec(\equiv_A^d)$. The number of equivalence classes $nec_d(T,\delta)$ of a decomposition (T,δ) is defined as $\max(nec(\equiv_A^d), nec(\equiv_{\overline{A}}^d))$ over all cuts $(A,\overline{A})$ of (T,δ).

Note that $N_A^1(X) = N(X) \cap \overline{A}$. It can then be observed that $|\mathcal{UN}(A)| = nec(\equiv_A^1)$ [16, Theorem 3.5.5] Also note that $X \equiv_A^d Y$ if and only if $N_A^d(X) = N_A^d(Y)$.

5.2 Bounds on the number of equivalence classes

We present a brief overview of the most relevant bounds that are currently known, for which we make use of a *twin class partition* of a graph.

▶ **Definition 15** (Twin class partition). Let $G = (V, E)$ be a graph of size n and let $A \subseteq V$. The *twin class partition* of A is a partition of A such that $\forall x, y \in A$, x and y are in the same partition class if and only if $N(x) \cap \overline{A} = N(y) \cap \overline{A}$. The number of partition classes of A is denoted by $ntc(A)$ and it holds that $ntc(A) \leq \min(n, 2^{\text{bool-dim}(A)})$ [2].

For all bounds listed below, let $G = (V, E)$ be a graph of size n and let d be a non-negative integer. Let $(A, \overline{A})$ be a cut induced by any node of a decomposition (T, δ) of G, and let $k = \text{bool-dim}(A) = nec(\equiv_A^1)$.

▶ **Lemma 16** ([3, Lemma 5]). $nec(\equiv_A^d) \leq 2^{d \cdot k^2}$.

▶ **Lemma 17** ([16, Lemma 5.2.2]). $nec(\equiv_A^d) \leq (d+1)^{\min(ntc(A), ntc(\overline{A}))}$.

▶ **Lemma 18.** $nec(\equiv_A^d) \leq ntc(A)^{d \cdot k}$.

By Lemma 16 we conclude that we can solve (σ, ρ) problems in $O^*(8^{dk^2})$. This shows that applications are more computationally expensive than using heuristics to find a decomposition.

6 Experiments

The experiments in this section are performed on a 64-bit Windows 7 computer, with a 3.40 GHz Intel Core i5-4670 CPU and 8GB of RAM. We implemented the algorithms using the C# programming language and compiled our programs using the *csc* compiler that comes with Visual Studio 12.0.[1]

6.1 Comparing Heuristics on random graphs

We will look at the performance of heuristics on randomly generated graphs, for which we used the Erdös-Rényi-model [4] to generate a fixed set of random graphs with varying edge probabilities.

In these experiments we start a heuristic once for each possible initial vertex, so n times in total. For the RelativeNeighborhood heuristic we select the best decomposition based upon the sum of the score function for all cuts, since computing all actual linear boolean-width values would take $O(n^3 \cdot 2^k)$ time, thereby removing the purpose of this polynomial time heuristic. For the set $Candidates$ we take $N^2(Left) \cap Right$, which avoids that we exclude certain optimal solutions, as opposed to Sharmin [10], who restricted this set to $N(Left) \cap Right$. However, this does not affect the results significantly.

We let the edge probability vary between 0.05 and 0.95 with steps of size 0.05. For each edge probability value, we generated 20 random graphs. The result per edge probability is taken to be the average boolean-width over these 20 graphs, which are shown in Figure 1. It

[1] Source code of our implementations can be found on `https://github.com/Chiel92/boolean-width` and `https://github.com/FrankvH/BooleanWidth`

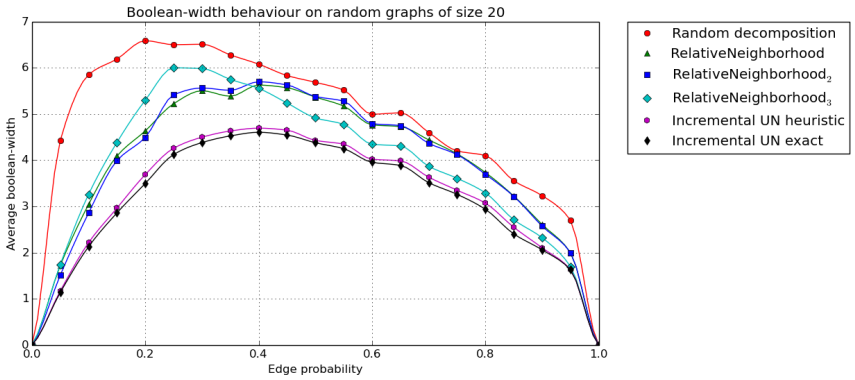

Figure 1 Performance of different heuristics on random generated graphs consisting of 20 vertices, with varying edge probabilities, in terms of linear boolean-width.

can be observed that the INCREMENTAL-UN-HEURISTIC procedure performs near optimal. Furthermore we see that the RELATIVENEIGHBORHOOD variants perform somewhere in between the optimal value and the value of random decompositions.

6.2 Comparing heuristics on real-world graphs

In order to get an idea of how the INCREMENTAL-UN-HEURISTIC compares to existing heuristics we compare them by both the boolean-width of the generated decomposition and the time needed for computation. We cannot compare the heuristics to the optimal solution, because computing an exact decomposition is not feasible on these graphs. The graphs that were used come from *Treewidthlib* [13], a collection of graphs that are used to benchmark algorithms using treewidth and related graph problems.

We ran the three different heuristics mentioned in Section 4 with *Candidates* = *Right* and with an additional two variations on the INCREMENTAL-UN-HEURISTIC (IUN) by varying the set of start vertices. The first variation, named 2-IUN, has two start vertices which are obtained through a single and double BFS respectively. The n-IUN heuristic uses all possible start vertices. For all other heuristics we obtained the start vertex through performing a double BFS. In Table 1 and 2 we present the results of our experiments.

It is expected that the IUN heuristic and LEASTCUTVALUE heuristic give the same linear boolean-width, since both these heuristics greedily select the vertex that minimizes the boolean dimension. The RELATIVENEIGHBORHOOD heuristic performs worse than all other heuristics in nearly all cases. While the difference might not seem very large, note that algorithms parameterized by boolean-width are exponential in the width of a decomposition. The 2-IUN heuristic outperforms IUN in three cases while n-IUN gives a better decomposition in 11 out of 13 cases, which shows that a good initial vertex is of great influence on the width of the decomposition.

Looking at the times displayed in Table 2 for computing each decomposition we see that the RELATIVENEIGHBORHOOD heuristic is significantly faster. This is to be expected because of the $O(n^3)$ time, compared to the exponential time for all other heuristics. The interesting comparison that we can make is the difference between the IUN heuristic and LEASTCUTVALUE heuristic. While both of these heuristics give the same decomposition, IUN is significantly faster. Additionally, even 2-IUN and n-IUN are often faster than the LEASTCUTVALUE heuristic.

Table 1 Linear boolean-width of the decompositions returned by different heuristics.

Graph	$\|V\|$	Edge Density	Relative	LeastCut	IUN	2-IUN	n-IUN
barley	48	0.11	5.70	5.91	5.91	4.70	4.58
pigs-pp	48	0.12	10.35	7.13	7.13	7.13	6.64
david	87	0.11	9.38	6.27	6.27	6.27	5.86
celar04-pp	114	0.08	11.67	7.27	7.27	7.27	7.27
1bkb-pp	127	0.18	16.81	9.98	9.98	9.53	9.53
miles1500	128	0.64	8.17	5.58	5.58	5.58	5.29
celar10-pp	133	0.07	10.32	11.95	11.95	7.64	6.91
munin2-pp	167	0.03	15.17	9.61	9.61	9.61	7.61
mulsol.i.5	186	0.23	7.55	5.29	5.29	5.29	3.58
zeroin.i.2	211	0.16	7.92	4.46	4.46	4.46	3.81
boblo	221	0.01	19.00	4.32	4.32	4.32	4.00
fpsol2.i-pp	233	0.40	5.58	6.07	6.07	5.78	4.81
munin4-wpp	271	0.02	13.04	9.27	9.27	9.27	7.61

Table 2 Time in seconds of the heuristics used to find linear boolean decompositions.

Graph	$\|V\|$	Edge Density	Relative	LeastCut	IUN	2-IUN	n-IUN
barley	48	0.11	< 0.01	0.18	0.01	0.02	0.16
pigs-pp	48	0.12	< 0.01	0.76	0.02	0.04	0.52
david	87	0.11	0.02	3.15	0.04	0.06	1.62
celar04-pp	114	0.08	0.04	5.73	0.14	0.23	9.85
1bkb-pp	127	0.18	0.06	198.05	1.14	4.18	107.32
miles1500	128	0.64	0.06	44.57	0.10	0.14	7.05
celar10-pp	133	0.07	0.06	8.93	1.96	4.72	18.43
munin2-pp	167	0.03	0.11	3.81	0.80	3.37	30.21
mulsol.i.5	186	0.23	0.09	37.88	0.13	0.27	8.80
zeroin.i.2	211	0.16	0.06	18.70	0.09	0.11	5.85
boblo	221	0.01	0.29	3.39	0.28	0.56	46.22
fpsol2.i-pp	233	0.40	0.18	189.11	0.36	0.74	56.63
munin4-wpp	271	0.02	0.61	57.87	1.98	6.66	367.37

6.3 Vertex subset experiments

We have used the linear decompositions given by the n-IUN heuristic to compute the size of the maximum induced matching (MIM) in a selection of graphs, of which the results are presented in Table 3. The maximum induced matching problem is defined as finding the largest $(\{1\}, \mathbb{N})$ set, with $d(\{1\}, \mathbb{N}) = 2$. The choice for the MIM problem is arbitrary, any vertex subset problem with $d = 2$ will have the same number of equivalence classes and therefore they all require the same time when computing a solution. We present the computed value of $nec_d(T, \delta)$, together with theoretical upperbounds. For $d = 2$ a tight upperbound in terms of boolean-width is not known. Note that we take the logarithm of each value, since we find this value easier to interpret and compare to other graph parameters. We let $UB_1 = 2^{d \cdot \text{boolw}^2}$, $UB_2 = (d+1)^{\min ntc}$ and $UB_3 = ntc^{d \cdot \text{boolw}}$, with $ntc = \max_{w \in T} ntc(\delta(w))$ and $\min ntc = \max_{w \in T} \min(ntc(\delta(w)), ntc(\overline{\delta(w)}))$.

The column *MIM* displays the size of the MIM in the graph, while the time column indicates the time needed to compute this set. Missing values for *nec* and MIM are caused

Table 3 Results of using the algorithm by Bui-Xuan et al. [3] for solving (σ, ρ) problems on graphs, using decompositions obtained through the n-IUN heuristic.

Graph	boolw	$\log_2(nec)$	$\log_2(UB_1)$	$\log_2(UB_2)$	$\log_2(UB_3)$	*MIM*	Time (s)
barley	4.58	7.00	42.04	12.68	27.51	22	3
pigs-pp	6.64	10.31	88.28	19.02	49.17	22	1147
david	5.86	9.37	68.63	22.19	44.61	34	919
celar04-pp	7.27	11.15	105.61	28.53	65.74	–	–
1bkb-pp	9.53	–	181.47	52.30	98.49	–	–
miles1500	5.29	9.30	55.87	34.87	49.69	8	4038
celar10-pp	6.91	10.34	95.41	25.36	59.70	50	10179
munin2-pp	7.61	11.82	115.97	19.02	54.60	–	–
mulsol.i.5	3.58	6.11	25.70	14.26	24.80	46	22
zeroin.i.2	3.81	6.58	28.99	20.60	28.18	30	59
boblo	4.00	6.17	32.00	9.51	20.68	148	41
fpsol2.i-pp	4.81	8.07	46.22	22.19	36.61	46	934
munin4-wpp	7.61	12.13	115.97	19.02	57.98	–	–

by a lack of internal memory, because of the $O^*(nec_d(T,\delta)^2)$ space requirement. One can immediately see that there is a large gap between the upperbound for nec_2 in terms of boolean-width and nec_2 itself. Another interesting observation we can make by looking at the graphs zeroin.i.2 and boblo, is that a lower boolean-width does not imply a lower nec_2. We even encountered this for decompositions of the same graph: for the graph barley we observed $\text{boolw}(T,\delta) = 4.58$ and $\text{boolw}(T',\delta') = 4.81$, while $\log_2(nec_2(T,\delta)) = 7.00$ and $\log_2(nec_2(T',\delta')) = 6.75$. This suggests that this upperbound does not justify minimizing nec_2 through boolean-width in practice.

7 Conclusion

We have presented a new heuristic and a new exact algorithm for finding linear boolean decompositions. The heuristic has a running time that is several orders of magnitude lower than the previous best heuristic and finds a decomposition in output sensitive time. This means that if a decomposition is not found within reasonable time, then the decomposition that would have been generated is not useful for practical algorithms. Running the new heuristic once for every possible starting vertex results in significantly better decompositions compared to existing heuristics.

We have seen that if $\text{lboolw}(T,\delta) < \text{lboolw}(T',\delta')$, then there is no guarantee that $nec(T,\delta) < nec(T',\delta')$. While in general it holds that minimizing boolean-width results in a low value of number of equivalence classes, we think that it can be worthwhile to focus on minimizing the nec_d instead of the boolean-width when solving vertex subset problems. However, the number of equivalence classes is not symmetric, i.e., for a cut $(A, \overline{A})$ $nec_d(A) \neq nec_d(\overline{A})$, which makes it harder to develop fast heuristics that focus on minimizing nec_d since we need to keep track of both the equivalence classes of A and $\overline{A}$.

Further research can be done in order to obtain even better heuristics and better upperbounds on both the linear boolean-width and boolean-width on graphs. For instance, combining properties of the Incremental-UN-heuristic and the RelativeNeighborhood heuristic might lead to better decompositions, as they make use of complementary features of a graph. Another approach for obtaining good decompositions could be a branch

and bound algorithm that makes us of trivial cases that are used in the heuristics. To decrease the time needed by the heuristics one can investigate reduction rules for linear boolean-width. While most reduction rules introduced by Sharmin [10] for boolean-width do not hold for linear boolean-width, they can still be used on a graph after which we can use our heuristic on the reduced graph. Although the resulting decomposition after reinserting the reduced vertices will not be linear, the asymptotic running time for applications does not increase [14]. Another topic of research is to compare the performance of vertex subset algorithms parameterized by boolean-width to algorithms parameterized by treewidth [15].

References

1 R. Belmonte and M. Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theoretical Computer Science*, 511:54–65, 2013. Exact and Parameterized Computation.

2 B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle. Boolean-width of graphs. In *IWPEC 2009*, volume 5917 of *LNCS*, pages 61–74. Springer, 2009.

3 B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theoretical Computer Science*, 511:66–76, 2013.

4 P. Erdös and A. Rényi. On random graphs. *Publicationes Mathematicae 6: 290–297*, 1959.

5 E. M. Hvidevold, S. Sharmin, J. A. Telle, and M. Vatshelle. Finding good decompositions for dynamic programming on dense graphs. In *IWPEC 2012*, volume 7112 of *LNCS*, pages 219–231. Springer, 2012.

6 K. H. Kim. *Boolean Matrix Theory and its Applications*. Marcel Dekker, 1982.

7 F. Manne and S. Sharmin. Efficient counting of maximal independent sets in sparse graphs. In *Experimental Algorithms*, volume 7933 of *LNCS*, pages 103–114. Springer, 2013.

8 Y. Rabinovich, J. A. Telle, and M. Vatshelle. Upper bounds on boolean-width with applications to exact algorithms. In *IWPEC 2013*, volume 8246 of *LNCS*, pages 308–320. Springer, 2013.

9 N. Robertson and P. D. Seymour. Graph minors. I. Excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983.

10 S. Sharmin. *Practical Aspects of the Graph Parameter Boolean-width*. PhD thesis, University of Bergen, Norway, 2014.

11 J. A. Telle. Complexity of domination-type problems in graphs. *Nordic Journal of Computing*, 1(1):157–171, 1994.

12 Ch. B. Ten Brinke, F. J. P. van Houten, and H. L. Bodlaender. Practical Algorithms for Linear Boolean-width. *ArXiv e-prints ArXiV:1509.07687*, 2015.

13 Treewidthlib. http://www.staff.science.uu.nl/∼bodla101/treewidthlib/. A benchmark for algorithms for treewidth and related graph problems.

14 F. J. P. van Houten. Experimental research and algorithmic improvements involving the graph parameter boolean-width. Master's thesis, Utrecht University, The Netherlands, 2015.

15 J. M. M. van Rooij, H. L. Bodlaender, and P. Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *Algorithms – ESA 2009*, volume 5757 of *LNCS*, pages 566–577. Springer, 2009.

16 M. Vatshelle. *New Width Parameters of Graphs*. PhD thesis, University of Bergen, Norway, 2012.