On Satisfiability Problems with a Linear Structure

Serge Gaspers[1,2], Christos Papadimitriou[3], Sigve Hortemo Sæther[4], and Jan Arne Telle[4]

1 **UNSW, Sydney, Australia**
sergeg@cse.unsw.edu.au
2 **Data61 (formerly NICTA), Sydney, Australia**
3 **UC Berkeley, USA**
christos@berkeley.edu
4 **University of Bergen, Norway**
sigve.sether@ii.uib.no, telle@ii.uib.no

Abstract

It was recently shown [18] that satisfiability is polynomially solvable when the incidence graph is an interval bipartite graph (an interval graph turned into a bipartite graph by omitting all edges within each partite set). Here we relax this condition in several directions: First, we show that it holds for k-interval bigraphs, bipartite graphs which can be converted to interval bipartite graphs by adding to each node of one side at most k edges; the same result holds for the counting and the weighted maximization version of satisfiability. Second, given two linear orders, one for the variables and one for the clauses, we show how to find, in polynomial time, the smallest k such that there is a k-interval bigraph compatible with these two orders. On the negative side we prove that, barring complexity collapses, no such extensions are possible for CSPs more general than satisfiability. We also show NP-hardness of recognizing 1-interval bigraphs.

1 Introduction

Constraint satisfaction problems (CSPs) such as satisfiability are both ubiquitous and difficult to solve. It is therefore essential to identify and exploit any special structure of instances that make CSPs susceptible to algorithmic techniques. One large class of such structured instances comprises CSPs whose constraints can be arranged in a linear manner, presumably reflecting temporal or spatial ordering of the real-life problem being modeled. A well known example is the *car sequencing* class of CSPs proposed by the French automobile manufacturer Renault in 2005 and reviewed in [21].

But defining what it means for a CSP to have "a linear structure" is not straightforward. The linear structure should be reflected in the incidence graph of the instance, but how exactly? Previous work has considered satisfiability instances with incidence graphs of bounded tree-width or bounded clique-width [5, 15, 19, 20, 23]. Instances that are in some sense close to efficiently solvable instances have been studied in terms of backdoors [7, 24], in particular for CNF formulas that have a small number of variables whose instantiations give formulas of bounded treewidth [8]. An important special case of bounded tree-width is bounded path-width, a measure of how path-like a graph is and a strong indication of linear structure. Bounded clique-width is a stronger notion, in which the graph's cliques have a linear structure.

Recently another direction for defining linear structure in CSPs was proposed, based in a time honored graph-theoretic conception of linear structure: *interval graphs,* the intersection graphs of intervals on the line. Interval graphs are a well-known class of graphs, going back to the 1950s, used to model temporal reasoning [9], e.g. in resource allocation and scheduling [1]. However, the incidence graphs we care about are bipartite, and the only connected interval graphs that are bipartite are trees. A bipartite version of interval graphs was introduced by Harary et al. in 1982 [11]: An *interval bigraph* is, informally, a bipartite graph[1] in which each vertex is associated with an interval, and there is an edge between two vertices *on different sides* if and only if the corresponding intervals intersect. Interval bigraphs form a natural and fairly rich class of bipartite graphs, containing, *e.g.,* all bipartite permutation graphs, which have been shown to have unbounded clique-width and thus also unbounded treewidth or pathwidth [2].

Interval bigraphs have been studied quite extensively, and several important facts are known about them. First, they can be recognized in polynomial time: In 1997 Müller gave an algorithm with running time $\mathcal{O}(|V||E|^6(|V|+|E|)\log|V|)$ [13], and a 2012 technical report [16] gives an algorithm with running time $\mathcal{O}(|V|(|E|+|V|))$. Importantly, Hell and Huang [12] gave in 2004 a useful alternative characterization of interval bigraphs as all bipartite graphs whose set of vertices can be ordered on the line so that the set of neighbors of each vertex coincides with an interval whose high end is the position of the vertex (see Lemma 2 below for the formal statement).

Interval bigraphs constitute a natural basis for identifying an important class of CSPs possessing a linear order: Define *an interval CSP* as a CSP whose variable-constraint incidence graph is an interval bigraph. In [18] a general dynamic programming approach to solving a class of CSPs was developed, and one consequence of that framework is that satisfiability — even weighted MAXSAT and #SAT — on interval CNF formulae with m clauses and n variables can be solved in time $\mathcal{O}(m^3(m+n))$ (stated as Theorem 3 below). See also the work of Brault-Baron et al [3] which sets out the wider context of the results of [18].

The present work is about extending Theorem 3 in several natural directions:

1. Many CSPs are not interval CSPs. Can the definition of interval CSPs be extended usefully, so that a limited number of "faults" in the interval structure of CSPs is tolerated by polynomial time algorithms?
2. In a variety of applications there is a natural linear ordering of both variables and constraints, but not of their union. One well-studied application is car sequencing [22], where sliding-window constraints [17] naturally come with an ordering of the variables and the constraints. In other temporal or scheduling settings, variables can be ordered according to the discrete timestep they are relevant to. Even when one is only given an ordering of the variables, one can greedily order the constraints; for example by their earliest variable in the variable-ordering or more domain-dependent methods. Under what circumstances is it possible to merge these two linear orders into one, so that the resulting bipartite graph is an interval bigraph?
3. If an order as in (2) above does not exist, can at least a merged order be found so the resulting bipartite graph is as close as possible (presumably in some algorithmically useful sense as in (1) above) to an interval bigraph?
4. Finally, to what extent can these algorithmic results be extended to CSPs beyond satisfiability?

In this paper we address and largely resolve these questions. In particular, our contributions are the following:

1. We define a useful measure of how much the incidence graph of a CSP instance differs from an interval bigraph: The smallest number k such that the incidence bigraph becomes

[1] We use "bigraph" and "bipartite graph" interchangeably.

interval if each constraint vertex of the bigraph has at most k edges added to it. Deciding if $k \leq 1$ is NP-hard (Theorem 6) but we show (Theorem 9) that given an ordering certifying a value of k such instances of satisfiability with m clauses and n variables can be solved in $\mathcal{O}(m^3 4^k(m+n))$ time. Ditto for MAXSAT and #SAT; the exponential dependence on k is, of course, expected.

2. We give a simple characterization of when two linear orders, one for constraints and one for variables, can be merged so that the resulting total order satisfies the Hell-Huang characterization of interval bigraphs.
3. We also show that, if no such merging is possible, we can find in polynomial time — through a greedy algorithm — the minimum k such that the incidence graph becomes interval with the addition of at most k edges to each constraint vertex. Hence, in the case of satisfiability, if this minimum is bounded then polynomial algorithms result.
4. Finally, we show that the approach in (1) above — which started us down this path — does not work for general CSPs, in that CSP satisfiability is intractable even when the incidence graph has the same favorable structure as in (1), with bounded k (Theorem 10).

Definitions and Background

Since we mostly deal with satisfiability, we denote our bipartite graphs as $G = (\texttt{cla}, \texttt{var}, E)$, where `cla` stands for clauses and `var` for variables.

▶ **Definition 1.** A bipartite graph $G = (\texttt{cla}, \texttt{var}, E)$ is an *interval bigraph* if every vertex can be assigned an interval on the real line such that for all $x \in \texttt{var}$ and $c \in \texttt{cla}$ we have $xc \in E$ if and only if the corresponding intervals intersect. A Boolean formula in conjunctive normal form (CNF) is called an *interval CNF formula* if the corresponding incidence graph (`cla` the clauses, `var` the variables, E the incidences) is an interval bigraph.

A most interesting alternative characterization of interval bigraphs by Hell and Huang [12] is stated here, expressed in terms of interval CNF formulas.

▶ **Lemma 2.** *[12] A CNF formula is an interval CNF formula if and only if its variables and clauses can be totally ordered (indicated by $<$) such that for any variable x appearing in a clause C:*

1. if x' is a variable and $x < x' < C$ then x' also appears in C, and

2. if C' is a clause and $C < C' < x$ then x also appears in C'.

We call an ordering of the variables and clauses of an interval CNF formula satisfying the lemma an *interval ordering*. Interval bigraphs can be recognized in polynomial time [13], see also [16].

2 k-interval Bigraphs

Recent work has articulated efficient algorithms in the dynamic programming style for interval CNF formulae.

▶ **Theorem 3.** *[18] Given an interval CNF formula on n variables and m clauses and an interval ordering of it, #SAT and weighted* MAXSAT *can be solved in time $\mathcal{O}(m^3(m+n))$.*

Combining Theorem 3 with the recognition algorithm of [13] gives the following:

▶ **Corollary 4.** *Given a CNF formula, it can be decided if it is an an interval CNF formula, and if so #SAT and weighted* MAXSAT *can be solved in polynomial time.*

We want to generalize this result to a larger class of formulae. To this end we introduce the following graph classes and formula classes, parametrized by $k \geq 1$.

▶ **Definition 5.** A bipartite graph $G = (\texttt{cla}, \texttt{var}, E)$ is a *k-interval bigraph* if we can add at most k edges to each vertex in `cla` such that the resulting bipartite graph is an interval bigraph. A CNF formula is called a *k-interval CNF formula* if its incidence graph (with clause vertices being `cla`) is a k-interval bigraph.

Note that 0-interval bigraphs are the interval bigraphs, and 1-interval bigraphs allow as many exceptions (added edges) as there are clauses. Unfortunately, the recognition problem for k-interval bigraphs becomes hard, already when $k = 1$. The proof is by reduction from the strongly NP-hard 3-Partition problem and is given in Section 5.

▶ **Theorem 6.** *Given a bipartite graph G and an integer k, deciding if G is a k-interval bigraph is NP-hard, even when $k = 1$.*

The alternative characterization of Lemma 2 can be extended to k-interval bigraphs.

▶ **Lemma 7.** *A CNF formula is a k-interval CNF formula if and only if its variables and clauses can be totally ordered such that for any clause C there are at most k variables x not appearing in C where either*
1. *a variable x' appears in C with $x' < x < C$, or*
2. *x appears in a clause C' with $C' < C < x$.*

Proof. The lemma follows directly from Definition 5 and Lemma 2. ◀

▶ **Definition 8.** For a k-interval CNF formula we call a total ordering of the kind guaranteed by Lemma 7 a k-interval ordering.

Our first algorithmic result is that, given a k-interval ordering of a k-interval CNF formula, #SAT and MaxSAT can be solved via a fixed-parameter tractable (FPT, see [4]) algorithm parameterized by k.

▶ **Theorem 9.** *Given a CNF formula and a k-interval ordering of it, we solve #SAT and weighted* MaxSAT *in time $\mathcal{O}(m^3 4^k (m + n))$.*

Proof. The full proof for #SAT and weighted MaxSAT is given in Section 4; here we give a straightforward construction establishing a weaker result for satisfiability only.

The basic observation is that the satisfiability of a CNF formula is not affected if a clause C is replaced by a particular set of clauses, defined next. Fix any set of $\ell \geq 0$ variables not occurring in C, and replace C with the 2^ℓ clauses of the form $(C \vee D_j) : j = 1, \ldots, 2^\ell$, where D_j ranges over the 2^ℓ possible clauses containing the ℓ variables. It is easy to see that a truth assignment satisfies the new formula if and only if it satisfied the original one. It is further clear that the satisfiability of the formula remains unaffected if all clauses are so replaced, for different sets of variables and $\ell \geq 0$. Finally, if a CNF formula is k-interval, then it has such an equivalent variant whose incidence graph is an interval bigraph. An FPT algorithm (albeit with running time $\mathcal{O}(m^3 8^k (m 2^k + n))$ instead of $\mathcal{O}(m^3 4^k (m + n))$) results. ◀

We next show that the k-interval structure is not helpful for general CSPs:

▶ **Theorem 10.** *Given a CSP instance I with variable-constraint incidence graph G and an interval bigraph G' obtained from G by adding at most k edges to each constraint vertex, deciding the satisfiability of I is $W[1]$-hard parameterized by k.*

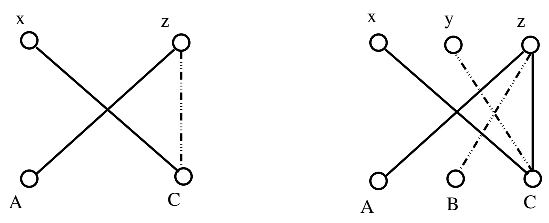

Figure 1 Obstructions to merging into an interval bigraph ordering: variables ordered $x < y < z$, clauses $A < B < C$, with solid lines indicating edges of the incidence graph and dotted lines indicating non-edges, with remaining possibilities being any combination of edges or non-edges.

Proof. Gottlob and Szeider [10] observed that CSP is $W[1]$-hard parameterized by the number of variables, recasting a W[1]-hardness proof for conjunctive queries in databases [14] to CSPs. The reduction is from clique, where for a graph $G = (V, E)$ and an integer parameter k, the question is whether there is a clique of size k in G, i.e., a set of k pairwise adjacent vertices. In the language of CSPs, their reduction construct a CSP with k variables $x_1, \ldots, x_k$, each with domain V. Intuitively, variable x_i represents the ith vertex of the clique we are looking for. For each pair of variables x_i, x_j with $i \neq j$, add a constraint with scope (x_i, x_j) whose constraint relation contains (u, v) iff $uv \in E$. Now, the CSP has a solution iff G has a clique of size k. Given a CSP instance with k variables, we can turn its incidence graph into an interval bigraph by adding all possible edges between variables and constraints. This creates a complete bipartite graph and adds at most k edges to each constraint vertex. ◀

3 Merging Linear Orders

Theorem 10 tells us that our ambition for new algorithmic results based on the concept of k-interval bigraph should be limited to CSPs of the satisfiability kind, while Theorem 6 suggests that the new concept of k-interval bigraph can only extend the class of solvable problems either in special cases, or indirectly, in specific contexts. In this section we derive an algorithmic result of the latter type.

Suppose that the real life situation modelled by the CNF formula has linearly ordered clauses, and linearly ordered variables, but there is no readily available linear order for both. That is, we assume the input comes with two linear orderings, one for the variables and one for the clauses. We wish to find the minimum value of k such that there exists a k-interval ordering compatible with both.

Problem: MERGING TO MINIMUM k-INTERVAL BIGRAPH ORDERING
Input: Bipartite graph $G = (\texttt{cla}, \texttt{var}, E)$, a total order of `cla`, and a total order of `var`
Output: The minimum k such that we can merge the two orders into a k-interval ordering of $\texttt{cla} \cup \texttt{var}$.

Consider first the case $k = 0$.

▶ **Lemma 11.** *If a formula is given with variable ordering, clause ordering, and incidences containing one of the obstructions in Figure 1, then it cannot be merged into an interval bigraph ordering.*

Proof. Consider the left-hand obstruction in Figure 1. We cannot insert z after C, since we get $A < C < z$ violating Condition 2 in Lemma 2. On the other hand, we cannot insert z before C, since we get $x < z < C$ violating Condition 1 in Lemma 2.

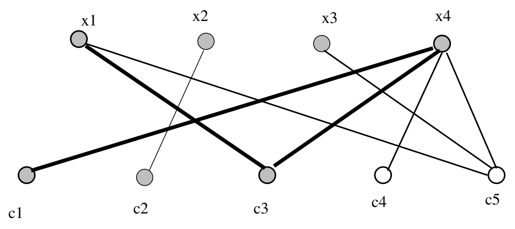

Figure 2 Consider the above input, with non-incidences indicated by non-edges. The bold edges and gray nodes show two overlapping obstructions as on the right side of Figure 1. Applying Lemma 2 these obstructions can be fixed in at least two ways: adding edge $c2x4$ by positioning $c3 < x2$; or adding edges $c3x2$ and $c3x3$ by positioning $c2 > x4$. In this last case a new obstruction appears, see Figure 3.

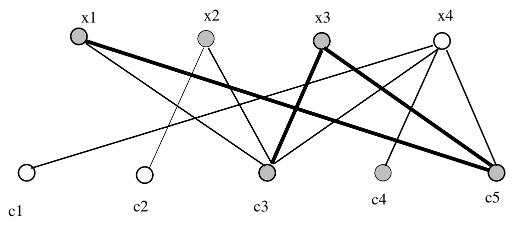

Figure 3 Assume we fixed the obstructions from Figure 2 by adding edges $c3x2$ and $c3x3$. We then get a new obstruction based in bold edges and gray nodes.

Consider the right-hand obstruction in Figure 1. We cannot insert z after B, since we get $A < B < z$ violating Condition 2 in Lemma 2. On the other hand, we cannot insert y before C, since we get $x < y < C$ violating Condition 1 in Lemma 2. Thus, since $B < C$ this leaves no place to insert z without violating Lemma 2. ◀

It turns out that, if there are no obstructions as in Figure 1 then MERGING TO MINIMUM k-INTERVAL BIGRAPH ORDERING has a solution with $k = 0$. Thus, for any instance where the solution has value $k > 0$ we can view the task as one of iteratively adding edges until the result has no obstruction as in Figure 1. On the face of it this is non-trivial, as there is more than one way of fixing an obstruction, with varying edge costs, and some ways may lead to a new obstruction appearing. For an example of this see Figures 2 and 3.

Nevertheless, a greedy approach will efficiently solve MERGING TO MINIMUM k-INTERVAL BIGRAPH ORDERING. Let us describe it. Assume the input ordering on variables and clauses is $x_1, ..., x_n$ and $c_1, ..., c_m$. All orderings we consider will be compatible with these input orderings. The greedy strategy works as follows. Start with $k = 0$ and consider clauses by decreasing index c_m, c_{m-1}, etc. Insert c_i among the variables in the highest possible position, below the position of c_{i+1}, that does not lead to more than k edges being added to c_i. If no such position exists then increase k and start all over again with c_m. The correctness of this strategy relies on the following observation.

▶ **Observation 12.** For any fixed position of c_i among the variables the number of edges we must add to clause c_i does not depend on where the other clauses are inserted, as long as $c_1, ..., c_{i-1}$ end up below c_i and $c_{i+1}, ..., c_m$ above c_i.

Proof. By Lemma 7 we must add to c_i exactly one edge for each variable x not appearing in c_i, where x satisfies one of the two conditions stated in Lemma 7. For the first condition note that C' can be any of $c_1, ..., c_{i-1}$ but no other clause. For the second condition note that it does not depend on any other clause, only on the position of c_i among the variables. ◀

In our greedy strategy, when deciding where to insert c_i the only restriction imposed on us by earlier decisions is that c_i must end up below the position of c_{i+1}. To allow the maximum degree of freedom we simply ensure that we have inserted c_{i+1} in the highest possible position. The pseudocode is in Figure 4.

```
Greedy Algorithm for merging to minimum $k$-interval bigraph ordering

input:   $G = (\texttt{cla}, \texttt{var}, E)$, orderings $\texttt{cla} = c_1, c_2, ..., c_m$ and $\texttt{var} = x_1, x_2, ..., x_n$
output:  minimum $k$ such that cla and var can be merged into a $k$-interval ordering

  $q := -1$;
  $success := false$;
  while not $success$
    $q := q + 1$;
    start with the ordering $x_1, x_2, ..., x_n$;
    for $i = m$ downto 1
        insert $c_i$ at the highest position, below $c_{i+1}$, where $EdgesAdded(c_i) \leq q$;
        if no such position exists for clause $c_i$ then break out of the for loop;
    if all clauses have been inserted then $success := true$;
  output $q$;

  $EdgesAdded(C)$:= number of variables satisfying one of the conditions of Lemma 7
```

Figure 4 Greedy Algorithm for merging to minimum k-interval bigraph ordering

▶ **Theorem 13.** *The Greedy Algorithm for* Merging to minimum k-interval bigraph ordering *is correct and can be implemented to run in time $\mathcal{O}(|E| \log k)$.*

Proof. Let us first argue for correctness. Consider an iteration of the inner loop that successfully found a position for clause c_i among the variables. For the current value of q it is not possible to insert c_i higher than this position without some c_j needing more than q edges added, for some $i \leq j \leq m$. This is in fact a loop invariant, as we inserted the clauses of higher index in the highest possible positions under exactly this constraint, and by Observation 12 their position does not influence the number of edges added to other clauses. Similarly, if for some c_i and current value of q we encounter 'no such position exists' then in any ordering of $\texttt{cla} \cup \texttt{var}$ compatible with the input orders there will be some $c_j, i \leq j \leq m$ which will need more than k edges added. Thus, when the algorithm successfully finds positions for all clauses then the current value of q is the correct answer.

Let us now argue for the running time. For the $\log k$ factor, rather than iterating on q until we succeed, we can search for the minimum k by what is known as galloping search, i.e. try q equal to 1, 2, 4, 8, etc until we succeed for an integer q, and then do binary search in the interval $[q/2..q]$. To decide on positions for the clauses in time $\mathcal{O}(|E|)$, for a fixed q, we need several program variables. The pseudocode is in Figure 5.

We maintain for each $x \in \texttt{var}$ the value $live(x)$ as the number of live clauses x appears in, where a live clause is one whose position has not been decided yet. Also, we maintain $livevar$ as the number of variables indexed higher than the current x_t and appearing in a live clause. Finally, $var(c_i)$ are the variables in clause c_i and $low(c_i)$ the index of its lowest indexed variable. The number of edges needed for c_i if inserted immediately after x_t is then

$$EdgesAdded(c_i) = livevar + t - low(c_i) - |var(c_i)|$$

This is so since we must add to c_i exactly one edge for each variable x not appearing in c_i, where x satisfies one of the two conditions stated in Lemma 7. The first condition counts

Deciding if we can merge to a q-interval ordering, for fixed q, in $\mathcal{O}(|E|)$ time

```
$\forall x \in \texttt{var}$: $live(x)$ := number of clauses $x$ appears in
$\forall c \in \texttt{cla}$: $var(c)$ := the set of variables in $c$
        $low(c) := i$, lowest $i$ with $x_i \in var(c)$
$livevar := 0$;
$t := n$;
start with the ordering $x_1, x_2, ..., x_n$;
for $i := m$ downto 1
  $inserted := false$;
  while not $inserted$          /* try to insert $c_i$ after $x_t$ */
      if $livevar + t - low(c_i) - |var(c_i)| \leq q$ then
        insert $c_i$ after $x_t$;
        $inserted := true$;
        $\forall x_j \in var(c_i) : live(x_j) := live(x_j) - 1$;
                  if $live(x_j) = 0$ and $j > t$ then $livevar := livevar - 1$;
      else if $t = 0$ then halt: 'no for this value of $q$';
      else $t := t - 1$; if $live(x_t) > 0$ then $livevar := livevar + 1$;
'yes for this value of $q$';
```

Figure 5 Deciding if we can merge to a q-interval ordering, for fixed q, in $\mathcal{O}(|E|)$ time

the number of variables indexed higher than the current x_t and appearing in some clause indexed lower than c_i, i.e. $livevar$, minus the number of variables in c_i of index higher than t. The second condition counts the number of variables strictly between $x_{low(c_i)}$ and x_{t+1}, i.e. $t - low(c_i)$, minus the number of variables in c_i of index t or less. Summing these two counts we get the above. ◀

4 Proof of Theorem 9

In this section we prove Theorem 9, namely that if we are given a CNF formula and a k-interval ordering of it, we can solve #SAT and weighted MAXSAT in time $\mathcal{O}(m^3 4^k (m+n))$. We do this by showing that the input has linear ps-width at most $m2^k + 1$ and applying the following result.

▶ **Theorem 14.** *[18] Given a CNF formula F with n variables* var *and m clauses* cla*, and a linear ordering of* cla$\cup$var *showing that F has linear* ps*-width at most p, we solve* #SAT *and weighted* MAXSAT *in time $\mathcal{O}(p^2 m(m+n))$.*

We need to clarify what is meant by the linear ps-width of a formula. We start with the related notion of ps-value of a CNF formula F on variables var and clauses cla. For an assignment τ of var, we denote by $\texttt{sat}(F,\tau)$ the inclusion maximal set $\mathcal{C} \subseteq \texttt{cla}$ so that each clause in $\mathcal{C}$ is satisfied by τ. Such a subset $\mathcal{C} \subseteq \texttt{cla}$ is called *projection satisfiable*. The ps-value of F is defined to be the number of projection satisfiable subsets of clauses, *i.e.* $|\texttt{PS}(F)|$, where

$$\texttt{PS}(F) = \{\texttt{sat}(F,\tau) : \tau \text{ is an assignment of } \texttt{var}\} \subseteq 2^{\texttt{cla}}.$$

Now, consider a linear ordering $e_1, e_2, ..., e_{n+m}$ of var $\cup$ cla. For any $1 \leq i \leq n+m$ we define two disjoint subformulas $F_1(i)$ and $F_2(i)$ crossing the cut between $\{e_1, ..., e_i\}$ and $\{e_{i+1}, ..., e_{n+m}\}$. We define $F_1(i)$ to be the subformula we get by removing from F all clauses not in $\{e_1, ..., e_i\}$ followed by removing from the remaining clauses each literal of a variable not in $\{e_{i+1}, ..., e_{n+m}\}$, and we define $F_2(i)$ vice-versa, as the subformula we get by removing from F all clauses not in $\{e_{i+1}, ..., e_{n+m}\}$ followed by removing from the remaining clauses each literal of a variable not in $\{e_1, ..., e_i\}$.

The ps-width of this linear ordering is defined to be the maximum ps-value over all the $2(n+m)$ subformulas $F_1(1), F_2(1), F_1(2), ..., F_2(n+m)$ that cross a cut of the ordering. The linear ps-width of F is defined to be the minimum ps-width of all linear orderings of $\mathtt{var} \cup \mathtt{cla}$.

Before giving the lemma that will prove Theorem 9 we state a useful result.

▶ **Lemma 15.** *[18] Any interval ordering of an interval CNF formula has* ps*-width no more than the number of its clauses plus one.*

▶ **Lemma 16.** *Let F be a k-interval CNF formula on m clauses. Then any k-interval ordering of it has* ps*-width at most $m2^k + 1$.*

Proof. Starting from F on m clauses and its k-interval ordering π we first construct an interval CNF formula F' having at most $m2^k$ clauses. Any clause C of F for which Lemma 7 prescribes $k' \leq k$ added edges from C to some k' variables, will be replaced in F' by a set of $2^{k'}$ clauses consisting of the clause C extended by all linear combinations of these k' variables. Note that F' is then an interval CNF formula with an interval ordering π' we get from π by naturally expanding a clause C in π to the $2^{k'}$ clauses, in any order, that replace C in F'.

Applying Lemma 15 all we need to finalize our proof is to show that the ps-width of the k-interval ordering π of F is no larger than the ps-width of the interval ordering π' of F'. To do this we must consider cuts of π.

Consider subformulas $F_1(i)$ and $F_2(i)$ of F crossing a cut of π. We show that for the corresponding cut in π' (i.e. we cut π' in the corresponding place without splitting any of the expanded set of clauses) the ps-values of the subformulas F'_1 and F'_2 of F' associated with this cut has ps-value no smaller than the ps-values of $F_1(i)$ and $F_2(i)$. That is $|\mathtt{PS}(F_1(i))| \leq |\mathtt{PS}(F'_1)|$ and $|\mathtt{PS}(F_2(i))| \leq |\mathtt{PS}(F'_2)|$. Note that the variables of $F_1(i)$ and F'_1 are the same, and similarly the variables of $F_2(i)$ and F'_2 are the same. W.l.o.g., we focus on $F_1(i)$ and F'_1, which we assume have variables $\mathtt{var}_1$.

We need to show that if two assignments a, b of $\mathtt{var}_1$ have $\mathtt{sat}(F_1(i), a) \neq \mathtt{sat}(F_1(i), b)$ then also $\mathtt{sat}(F'_1, a) \neq \mathtt{sat}(F'_1, b)$. W.l.o.g., assume some clause $C \in \mathtt{sat}(F_1(i), a)$ but $C \notin \mathtt{sat}(F_1(i), b)$. We show that we can find a clause C' that distinguishes a and b in F'_1 as well. Clause C of $F_1(i)$ comes from an original clause (possibly larger, since C lives across a cut) in F. Assume this original clause was expanded in F' to $2^{k'}$ clauses, for some $k' \leq k$, by extending it with all linear combinations of the k' new variables. Depending on which variables are on the other side of the cut the clause C of $F_1(i)$ has been expanded to a set of $2^{k''}$, for some $k'' \leq k'$, clauses in F'_1, still consisting of all linear combinations of the k'' variables not in C. Since a satisfies C and C is a part of all these expanded clauses we have assignment a satisfying all of them. Since b does not satisfy C there will be exactly one of these $2^{k''}$ clauses that are not satisfied by b, namely the one where the linear combination of the new variables is falsified by assignment b. This means that $\mathtt{sat}(F'_1, a) \neq \mathtt{sat}(F'_1, b)$.

Thus the ps-width of the k-interval ordering of F is no more than the ps-width of the interval ordering of F' and we are done. ◀

Combining Theorem 14 with Lemma 16 we arrive at Theorem 9. Combining with Theorem 13 we get the following.

▶ **Corollary 17.** *Given a CNF formula and two total orderings, one for its m clauses and one for its n variables, we can in polynomial time find the minimum k such that these two orders can be merged into a k-interval ordering and then solve #SAT and MaxSAT in time $\mathcal{O}(m^3 4^k(m+n))$.*

5 Proof of Theorem 6

In this section we prove Theorem 6, namely that it is NP-hard to recognize k-interval bigraphs, already for $k = 1$.

Proof. We give a polynomial time reduction from the 3-Partition problem, which is strongly NP-hard [6]. Given an integer b, a set A of $3n$ elements, and a positive integer $s(a)$ for each $a \in A$ such that $b/4 < s(a) < b/2$ for each $a \in A$ and $\sum_{a\in A} s(a) = n\cdot b$, the question is whether A can be partitioned into disjoint sets $A_1, \dots, A_n$ such that $\sum_{a\in A_i} s(a) = b$ for each $i \in \{1, \dots, n\}$.

For a 3-Partition instance (b, A, s), we construct an instance $G = (V, E)$ for the 1-interval bigraph recognition problem as follows. We assume, w.l.o.g., that $b \geq 4$, and therefore, $s(a) > 1$ for each $a \in A$.

We add a set of *slot* vertices $S = \bigcup_{i=1}^{n} S_i$ with $S_i = \{s_{i,1}, \dots, s_{i,b+1}\}$. For all i, j with $1 \leq i \leq n$ and $1 \leq j \leq b$ we add a vertex $\ell_{i,j}$ that is adjacent to both $s_{i,j}$ and $s_{i,j+1}$, so that $(s_{i,1}, \ell_{i,1}, s_{i,2}, \ell_{i,2}, \dots, s_{i,b}, \ell_{i,b}, s_{i,b+1})$ is a path for each $i \in \{1, \dots, n\}$.

For each $i \in \{1, \dots, n-1\}$ we add a *delimiter* vertex s_i^d, and two vertices $\ell_i^{d,1}$ and $\ell_i^{d,2}$. We make $\ell_i^{d,1}$ adjacent to $s_{i,b}$, $s_{i,b+1}$, s_i^d, and $s_{i+1,1}$ and we make $\ell_i^{d,2}$ adjacent to $s_{i,b+1}$, s_i^d, $s_{i+1,1}$, and $s_{i+1,2}$. The set of delimiter vertices is $D = \bigcup_{i=1}^{n-1} \{s_i^d\}$.

We add a *track* vertex t that is adjacent to each vertex in $S \cup D \setminus \{s_1^d\}$.

We add *left anchor* vertices a^l, $\ell^{a,l}$, and make $\ell^{a,l}$ adjacent to a^l, $s_{1,1}$, and $s_{1,2}$. Symmetrically, we add *right anchor* vertices a^r, $\ell^{a,r}$, and make $\ell^{a,r}$ adjacent to a^r, $s_{n,b+1}$, and $s_{n,b}$. See Figure 6 for an illustration of the graph constructed so far.

For each element $a \in A$, we add a *numeral* gadget which is obtained from a path on $2 \cdot s(a) + 1$ new vertices $(\ell_{a,0}^n, n_{a,1}, \ell_{a,1}^n, \dots, n_{a,s(a)-1}, \ell_{a,s(a)-1}^n, n_{a,s(a)}, \ell_{a,s(a)}^n)$ and the track vertex t is made adjacent to $n_{a,1}, \dots, n_{a,s(a)}$. See Figure 7 for an illustration of a numeral gadget.

We will now show that (b, A, s) is a Yes-instance for 3-Partition if and only if G is a 1-interval bigraph. For the forward direction, consider a solution $A_1, \dots, A_n$ to the 3-Partition instance. We construct an interval representation following the scheme outlined in Figure 6, which is missing the numeral gadgets. Now, for each $A_i = \{x, y, z\}$, we can intersperse the intervals $s_{i,1}, \dots, s_{i,s(x)+1}$ with the intervals $n_{x,1}, \dots, n_{x,s(x)}$, intersperse the intervals $s_{i,s(x)+1}, \dots, s_{i,s(x)+s(y)+1}$ with the intervals $n_{y,1}, \dots, n_{y,s(y)}$, and intersperse the intervals $s_{i,s(x)+s(y)+1}, \dots, s_{i,b+1}$ with the intervals $n_{z,1}, \dots, n_{z,s(z)}$. In this way, each vertex $\ell_{i,j}$, $1 \leq i \leq n$, $1 \leq j \leq b$, is non-adjacent to exactly one vertex (from $\{n_{a,1}, \dots, n_{a,s(a)} : a \in A\}$) whose corresponding intervals overlap, and each vertex $\ell_{a,j}^n$, $a \in A$, $1 \leq j \leq s(a) - 1$, is non-adjacent to exactly one vertex (from $\{s_{i,2}, \dots, s_{i,b} : 1 \leq i \leq n\}$) whose corresponding intervals overlap. This certifies that G is a 1-interval bigraph.

For the backward direction, we observe that our construction enforces the rigid structure from Figure 6. Intuitively, for each $i \in \{1, \dots, n\}$, the vertices $\ell_{i,j}$ enforce an ordering of the intervals corresponding to the vertices in S_i, and the delimiters glue the different sections of S_i vertices together in a linear fashion. Observe that between two vertices $s_{i,j}$ and $s_{i,j+1}$, we can still insert one vertex if it is adjacent to t, and we exploit this property to intersperse the numerals. The anchor vertices are used to stretch the structure of the slot

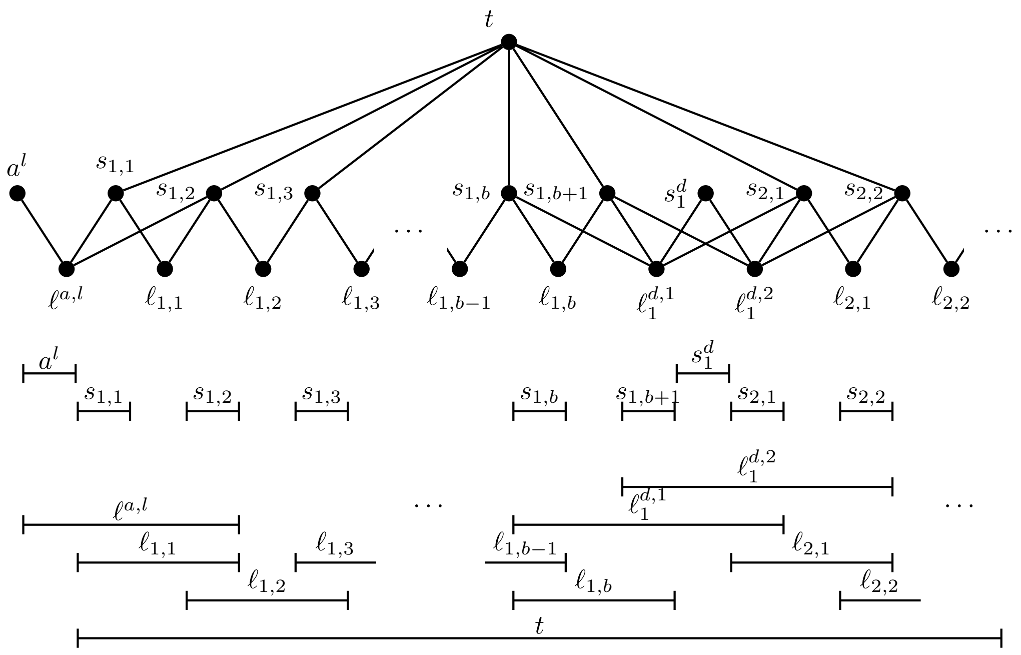

Figure 6 A part of the graph constructed by our reduction and a corresponding 1-interval representation formed by the all the vertices except the numeral gadgets. The top two rows of intervals correspond to the vertices in one partite set of the bipartition and the bottom rows to vertices in the other partite set.

vertices beyond the left and the right of the track t. This ensures then that the numerals need to be interspersed with the slots. Since there are no elements $a \in A$ with $s(a) = 1$, it is also not possible for a numeral gadget to intersperse a section S_i of slot vertices before $s_{i,1}$ or after $s_{i,b+1}$. In addition, the delimiters ensure that numerals do not straddle different S_i's. Therefore, we can obtain a solution to the 3-PARTITION instance by setting A_i to the elements from A that we used to construct the numeral gadgets that are interspersed with the slots in S_i. ◀

6 Discussion

The algorithmic challenge of CNF satisfiability and constraint satisfaction is central in both computational theory and practice, and new angles of attack to these age-old problems keep emerging. Here we focused on instances which possess a linear structure, and we proposed a new approach to dealing with local departures from such structure, as well as for deducing linear structure from partial evidence; we also identified complexity obstacles to fully exploiting and extending our approach. Our work raises several questions:

- What if only one side of the bipartite incidence graph is ordered? Say we are given an ordering of variables and asked if the clauses can be inserted so as to yield a k-interval ordering. For the case $k = 0$ we can use the obstructions in Figure 1 to guide us towards a linear ordering also of the clauses, e.g., for a pair of clauses A, C with two variables $x < z$ where xC, zA are edges and zC is a non-edge we must place C before A. We believe such an approach should solve the $k = 0$ case in polynomial time, but we are less optimistic about the general case of minimizing k.

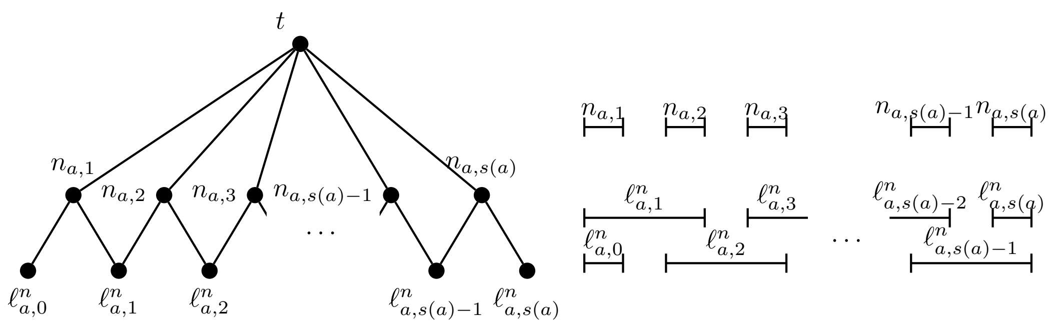
t
$n_{a,1}$
$n_{a,2}$
$n_{a,3}$
$n_{a,s(a)-1}$
$n_{a,s(a)}$
$\ell^n_{a,0}$
$\ell^n_{a,1}$
$\ell^n_{a,2}$
$\ell^n_{a,s(a)-1}$
$\ell^n_{a,s(a)}$
$n_{a,1}$ $n_{a,2}$ $n_{a,3}$ $n_{a,s(a)-1}$ $n_{a,s(a)}$
$\ell^n_{a,1}$ $\ell^n_{a,3}$ $\ell^n_{a,s(a)-2}$ $\ell^n_{a,s(a)}$
$\ell^n_{a,0}$ $\ell^n_{a,2}$ $\ell^n_{a,s(a)-1}$

Figure 7 A numeral gadget for element $a \in A$.

- What if we are given a partial order, with some special properties, on variables and clauses? Note that already the approach for $k = 0$ hinted at above could yield a situation with a linear order on variables and a partial order on clauses.
- For which industrial CNF instances can we find k-interval orderings for low values of k? Our greedy algorithm for merging two linear orders to a minimum k-interval ordering is practical and can be applied to large instances in the SAT corpora. In light of the hardness result for recognizing 1-interval bigraphs, heuristics or domain expertise could be used to generate orders for clauses and variables, when they are not already given.
- Which other classes of interval bigraph CSP instances can be solved efficiently? Our hardness result is for general CSPs with large domains. For CSPs with Boolean domains we can show a similar hardness result albeit not for k-interval bigraph instances, instead for a different notion of "imperfection" where we are given k pairs of clause vertices in the incidence graph such that merging each such pair results in an interval bigraph.

Acknowledgements This work was partially supported by a grant from the Peder Sather Center at UC Berkeley. Serge Gaspers is the recipient of an Australian Research Council (ARC) Future Fellowship (FT140100048) and acknowledges support under the ARC's Discovery Projects funding scheme (DP150101134). NICTA is funded by the Australian Government through the Department of Communications and the ARC through the ICT Centre of Excellence Program.

References

1 Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph (Seffi) Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 735–744, New York, NY, USA, 2000. ACM.

2 Andreas Brandstädt and Vadim V. Lozin. On the linear structure and clique-width of bipartite permutation graphs. *Ars Comb.*, 67, 2003.

3 Johann Brault-Baron, Florent Capelli, and Stefan Mengel. Understanding model counting for beta-acyclic cnf-formulas. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, pages 143–156, 2015.

4 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity.* Monographs in Computer Science. Springer, 1999.

5 Eldar Fischer, Johann A. Makowsky, and Elena V. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Applied Mathematics*, 156(4):511–529, 2008.

6 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

7 Serge Gaspers and Stefan Szeider. Backdoors to satisfaction. In Hans L. Bodlaender, Rod Downey, Fedor V. Fomin, and Dániel Marx, editors, *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*, pages 287–317. Springer, 2012.

8 Serge Gaspers and Stefan Szeider. Strong backdoors to bounded treewidth SAT. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 489–498. IEEE Computer Society, 2013.

9 Martin Charles Golumbic and Ron Shamir. Complexity and algorithms for reasoning about time: A graph-theoretic approach. *J. ACM*, 40(5):1108–1133, November 1993.

10 Georg Gottlob and Stefan Szeider. Fixed-parameter algorithms for artificial intelligence, constraint satisfaction and database problems. *Comput. J.*, 51(3):303–325, 2008.

11 Frank Harary, Jerald A Kabell, and Frederick R McMorris. Bipartite intersection graphs. *Commentationes Mathematicae Universitatis Carolinae*, 23(4):739–745, 1982.

12 Pavol Hell and Jing Huang. Interval bigraphs and circular arc graphs. *Journal of Graph Theory*, 46(4):313–327, 2004.

13 Haiko Müller. Recognizing interval digraphs and interval bigraphs in polynomial time. *Discrete Applied Mathematics*, 78(1-3):189–205, 1997.

14 Christos H. Papadimitriou and Mihalis Yannakakis. On the complexity of database queries. *Journal of Computer and System Sciences*, 58(3):407 – 427, 1999.

15 Daniël Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model counting for CNF formulas of bounded modular treewidth. In Natacha Portier and Thomas Wilke, editors, *STACS*, volume 20 of *LIPIcs*, pages 55–66. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

16 Arash Rafiey. Recognizing interval bigraphs by forbidden patterns. *CoRR*, abs/1211.2662, 2012.

17 Jean-Charles Régin and Jean-Francois Puget. A filtering algorithm for global sequencing constraints. In *Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming (CP 1997)*, volume 1330 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 1997.

18 Sigve Hortemo Sæther, Jan Arne Telle, and Martin Vatshelle. Solving #SAT and MAXSAT by dynamic programming. *J. Artif. Intell. Res. (JAIR)*, 54:59–82, 2015.

19 Marko Samer and Stefan Szeider. Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1):50–64, 2010.

20 Friedrich Slivovsky and Stefan Szeider. Model counting for formulas of bounded clique-width. In Leizhen Cai, Siu-Wing Cheng, and Tak Wah Lam, editors, *ISAAC*, volume 8283 of *Lecture Notes in Computer Science*, pages 677–687. Springer, 2013.

21 Christine Solnon, Van-Dat Cung, Alain Nguyen, and Christian Artigues. The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge problem. *European Journal of Operational Research*, 191(3):912–927, 2008.

22 Christine Solnon, Van-Dat Cung, Alain Nguyen, and Christian Artigues. The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the roadef'2005 challenge problem. *European Journal of Operational Research*, 191(3):912–927, 2008.

23 Stefan Szeider. On fixed-parameter tractable parameterizations of SAT. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT 2003*, volume 2919 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2003.

24 Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors to typical case complexity. In Georg Gottlob and Toby Walsh, editors, *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 1173–1178. Morgan Kaufmann, 2003.