

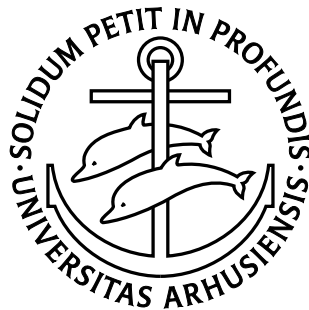
# Exact Algorithms for Satisfiability Problems and Graph Theoretical Problems

Bolette Ammitzbøll Madsen

---

---

## Progress Report



Department of Computer Science  
University of Aarhus  
Denmark



# Exact Algorithms for Satisfiability Problems and Graph Theoretical Problems

A Progress Report  
Presented to the Faculty of Science  
of the University of Aarhus  
in Partial Fulfilment of the Requirements for the  
PhD Degree

by  
Bolette Ammitzbøll Madsen  
15th November 2002



# Abstract

In this progress report I present an exact algorithm for the NP-complete problem X3SAT: Exact satisfiability with at most three variables in each clause. The algorithm builds on top of the Davis-Putnam algorithm and achieves the best known time complexity of  $\mathcal{O}^*(2^{0.1626n})$  for X3SAT.

I also present upper and lower bounds on the maximum number of maximal bipartite subgraphs in a graph. These are results from the technical report [14]. Along with the upper bound, I also present an enumeration algorithm for maximal bipartite subgraphs with time complexity  $\mathcal{O}^*(2^{0.8963n})$ . It was hoped that this algorithm could be used for an exact algorithm for the NP-complete problem  $k$ -COLOURING, however the upper bound is not strong enough.

I discuss plans for future work on exact algorithms, including an algorithm for general XSAT based on the same ideas as the X3SAT algorithm.



# Contents

<b>1</b>	<b>Introduction</b>	<b>9</b>
<b>2</b>	<b>Satisfiability Problems</b>	<b>13</b>
2.1	Introduction to Satisfiability . . . . .	13
2.1.1	The Davis-Putnam Algorithm . . . . .	14
2.2	Exact Satisfiability . . . . .	17
2.2.1	An exact algorithm for X3SAT . . . . .	19
<b>3</b>	<b>Graph Theoretical Problems</b>	<b>25</b>
3.1	Maximal Independent Sets . . . . .	26
3.1.1	An Enumeration Algorithm . . . . .	26
3.1.2	Applications . . . . .	27
3.2	Maximal Bipartite Subgraphs . . . . .	29
3.2.1	An Enumeration Algorithm . . . . .	29
3.2.2	Applications . . . . .	32
<b>4</b>	<b>Future Work</b>	<b>35</b>
	<b>Bibliography</b>	<b>39</b>



# Chapter 1

## Introduction

This progress report presents part of the work I have done in part A of my PhD and plans for work I would like to do in part B. Some exact algorithms for well known NP-complete problems are presented. My own contributions lie in the algorithm for X3SAT and the bounds on the number of maximal bipartite subgraphs in a graph. All the results are joint work with Jesper Makholm Byskov and Bjarke Skjerna.

**Types of Algorithms** Most of the algorithms I have looked at are exact (exponential) algorithms for NP-complete problems. An exact algorithm always gives the correct (exact) answer, and one can prove an exact bound on the time complexity. Most exact algorithms, while taking exponential time, only use polynomial space.

I will also present two enumeration algorithms, which enumerate and output an exponential number of results. However as only the output uses exponential space, they can be modified to algorithms for solving certain NP-complete problems using only polynomial space.

**Complexity** I will use the notation  $\mathcal{O}^*(2^{\alpha n})$  for all functions within a polynomial factor of  $2^{\alpha n}$ . It might seem futile to improve one exponential time bound to another exponential time bound, but notice that improving the time bound from for instance  $\mathcal{O}^*(2^n)$  to  $\mathcal{O}^*(2^{n/2})$  actually means that we can handle problems roughly twice as large as before within the same time.

**Estimating Tree Sizes** There are of course different types of exact algorithms for NP-complete problems. Many of them, however, boil down to some kind of branching algorithm. This is also the case for the algorithms I have been working on, so I will briefly describe how to compute the time complexity of such algorithms.

A branching algorithm corresponds to a tree, where the recursion formula for the time complexity  $T(n) = T(n - t_1) + \dots + T(n - t_r)$  gives the size of the whole tree by adding up the sizes of the subtrees. We say that each branching of the algorithm has a *branching vector* of  $(\mathbf{t}_1, \dots, \mathbf{t}_r)$ .

For exhaustive search we get a branching vector of  $(\mathbf{1}, \mathbf{1})$ , and we know that the time complexity is  $\mathcal{O}^*(2^n)$ . But what if we for instance have an algorithm with the worst case branching vector  $(\mathbf{2}, \mathbf{3})$ ?

We define the characteristic polynomial associated with the branching vector  $(\mathbf{t}_1, \dots, \mathbf{t}_r)$  to be  $p(x) = 1 - \sum_{i=1}^r x^{-t_i}$ , and it can be shown that  $p(x)$  has a unique positive real root  $\tau$ . The number of leaves of the tree is bounded by  $\tau_{max}^n$  (see for example [12]), where  $\tau_{max}$  is the maximum over all  $\tau$ 's in the tree (and  $\tau = 1$ , when we do not branch). As the size of the tree is bounded by a polynomial times the number of leaves this also gives us a bound on the tree size in  $\mathcal{O}^*$  notation, and therefore on the running time of the algorithm, as we only spend polynomial time in each node in the tree.

Returning to our example of branching vector  $(\mathbf{2}, \mathbf{3})$ , we see that the characteristic polynomial is  $p(x) = 1 - 1/x^2 - 1/x^3$ . The real positive root of this polynomial is approximately 1.3247, and our algorithm therefore has worst case time complexity  $\mathcal{O}^*(1.3247^n) \approx \mathcal{O}^*(2^{0.40569n})$ .

**Outline** First I will introduce various satisfiability problems and take a look at the famous Davis-Putnam algorithm, as this illustrates the idea of (exact) branching algorithms and is also the backbone of the algorithms for XSAT and X3SAT, which Jesper Makholm Byskov, Bjarke Skjernaa and I are currently writing a paper about. I then present the algorithm for X3SAT, which gives the currently best known time complexity for this problem.

Next I look at an enumeration algorithm for maximal independent sets, which gives rise to the best known bounds for 4-COLOURING and 5-COLOURING, and also to an enumeration algorithm for maximal bipartite subgraphs. The enumeration algorithm and the upper and lower bounds on the maximum number of maximal bipartite subgraphs in a graph are results from the technical report [14], which is also joint work with Jesper Makholm Byskov

and Bjarke Skjerna. The paper has been submitted to Journal of Graph Theory.

Finally I will discuss some plans for my future work on part B. I would like to continue to work on exact algorithms, and I suggest that there are research possibilities both within traditional exact algorithms and parameterised algorithms.



# Chapter 2

## Satisfiability Problems

A Boolean variable can take either of the two truth values **true** or **false**. A (quantifier free) Boolean formula  $F$  over the Boolean variables  $x_1, \dots, x_n$  combines the variables using the Boolean connectives  $\vee$  (or/disjunction),  $\wedge$  (and/conjunction) and  $\neg$  (not/negation).

A truth assignment assigns either **true** or **false** to each of the variables in  $F$  resulting in a Boolean expression, which we can evaluate. Some formulas are tautologies, which means that they are true under any assignment, some are unsatisfiable under any assignment and some can be satisfied by certain assignments.

A literal is either a variable  $x$  or the negation of a variable  $\neg x$  (also written  $\bar{x}$ ). A Boolean formula is in conjunctive normal form (CNF) if  $F = \bigwedge_{i=1}^m C_i$ , where each  $C_i$  (called a clause) is a disjunction of literals. Any Boolean formula is equivalent to a formula in CNF.

In a Boolean formula, let  $n$  denote the number of variables,  $m$  the number of clauses and  $l$  the number of literals.

### 2.1 Introduction to Satisfiability

First let us define the satisfiability problem or just SAT:

**Definition 1.** *The SAT problem is: Given a Boolean formula in conjunctive normal form, is it satisfiable?*

SAT was the first problem to be proven NP-complete by Cook in 1971 [4]. Cook also proved that  $k$ SAT (each clause has at most  $k$  literals) is NP-complete for  $k \geq 3$ .

A lot of related problems are also NP-complete. In 1978 Schaefer proved that NAESAT (a clause is satisfied if it has at least one true and one false literal), XSAT (a clause is satisfied if it has exactly one true literal) and also NAE $k$ SAT and X $k$ SAT ( $k \geq 3$ ) are NP-complete [21].

### 2.1.1 The Davis-Putnam Algorithm

We can solve an instance of the SAT problem by simply testing all  $2^n$  possible assignments, giving us a time complexity of  $\mathcal{O}^*(2^n)$ . However we would like to do better. If we insist on the number of variables as the input parameter, we do not know how to get a better bound. However, if we instead allow the number of clauses or literals as the parameter, we do.

Let  $F[x \leftarrow true]$  ( $F[x \leftarrow false]$ ) be the formula  $F$ , where the variable  $x$  is replaced with true (false). This algorithm was presented already in 1962 [6].

```

boolean DavisPutnam(CNFformula  $F$ )
   $F := \text{Reduce}(F)$ ;
  if  $F$  does not contain any variables then
    return Evaluate( $F$ )
  else
    Choose a variable  $x$  in  $F$ ;
     $F_0 := F[x \leftarrow \text{false}]$ ;
    if DavisPutnam( $F_0$ )=true then
      return true
    else
       $F_1 := F[x \leftarrow \text{true}]$ ;
      return DavisPutnam( $F_1$ )
  fi
fi

```

**Reductions** Now if 'Reduce' did nothing this would simply give us the algorithm testing all possible assignments. What 'Reduce' does is: While any of the following simple reduction rules can be applied, reduce  $F$  by the first applicable rule.

$$(x \vee \bar{x} \vee C) \wedge F' \rightarrow (\text{true} \vee C) \wedge F' \quad (2.1)$$

$$(x \vee x \vee C) \wedge F' \rightarrow (x \vee C) \wedge F' \quad (2.2)$$

$$C \wedge_{C \subseteq C'} C' \wedge F' \rightarrow C \wedge F' \quad (2.3)$$

$$(\text{true} \vee C) \wedge F' \rightarrow F' \quad (2.4)$$

$$(\text{true} \vee C) \rightarrow \text{true} \quad (2.5)$$

$$(\text{false} \vee C) \wedge F' \rightarrow C \wedge F' \quad (2.6)$$

$$(\text{false}) \wedge F' \rightarrow \text{false} \quad (2.7)$$

$$(x) \wedge F' \rightarrow F'[x := \text{true}] \quad (2.8)$$

$$(\bar{x}) \wedge F' \rightarrow F'[x := \text{false}] \quad (2.9)$$

$$x \text{ occurs only unnegated in } F' \rightarrow F'[x := \text{true}] \quad (2.10)$$

$$x \text{ occurs only negated in } F' \rightarrow F'[x := \text{false}] \quad (2.11)$$

All of these rules obey that if  $G$  is the Boolean formula that  $\text{Reduce}(F)$  returns, then  $G$  is satisfiable if and only if  $F$  is satisfiable. This is quite easy to see. It is also easy to see, that 'Reduce' runs in polynomial time, as checking or applying a rule takes time polynomial in the length of the formula, and each rule reduces either the number of literals (a constant is also counted as a literal) or the number of variables, thus we can do at most  $l + n$  reductions.

The Davis-Putnam procedure using these reduction rules runs in time at most  $\mathcal{O}^*(2^m)$ , as at least one clause is removed in each call to 'Reduce'. Notice that reduction rules 2.10 and 2.11 assures, that the variable  $x$ , on which we branch, occurs both unnegated and negated. Thus both setting  $x$  to true and to false, will set some literal to true, and then reduction rule 2.4 (or 2.5) will remove a clause.

**Resolution** The resolution principle is the following ( $x$  is a  $(p, q)$ -occurrence, meaning that  $x$  occurs  $p$  times unnegated and  $q$  times negated):

$$\bigwedge_{1 \leq i \leq p} (x \vee C_i) \wedge \bigwedge_{1 \leq j \leq q} (\bar{x} \vee C'_j) \wedge F' \rightarrow \bigwedge_{1 \leq i \leq p, 1 \leq j \leq q} (C_i \vee C'_j) \wedge F'$$

This rule also conserves satisfiability. Note that the resolution principle produces  $p \cdot q$  clauses from  $p + q$ . Our new reduction rule is to apply resolution when there is an  $x$  for which  $p \cdot q \leq p + q$ .

'Reduce' is still polynomial, even though it is a bit harder to see now, as the resolution rule may increase the number of literals. Note that resolution removes a variable, and therefore can be used at most  $n$  times. Now the two first rules assures us, that any clause can have at most  $n$  literals after 'Reduce', thus the number of literals can be at most  $n \cdot m$ , and as all the other rules reduces the number of literals or variables strictly, we have at most  $n^2 m$  iterations.

**Theorem 1.** *Davis-Putnam using resolution solves SAT in time at most  $\mathcal{O}^*(2^{0.40569m})$ .*

*Proof.* We look at how many clauses is removed in each branch. We know that for any variable  $x$  occurring  $p$  times unnegated and  $q$  times negated, we have  $p \cdot q \geq p + q$ , since it would otherwise have been removed by resolution. This implies that  $\min(p, q) \geq 2$  and  $\max(p, q) \geq 3$ .

Now when we branch on  $x$ , in one branch we set  $x$  to true, removing  $p$  clauses, and in the other branch we set  $x$  to false, removing  $q$  clauses. This gives us a branching vector of at least  $(2, 3)$ , which gives us the promised running time of  $\mathcal{O}^*(2^{0.40569m})$ .  $\square$

**Improvements** The Davis-Putnam algorithm has been improved a number of times. One of the well-known extensions is the Monien-Speckenmeyer algorithm [15] from 1980, which has a running time of  $\mathcal{O}^*(2^{m/3})$ . The best known bound using the Davis-Putnam pattern and the number of clauses as parameter was obtained by Hirsch in [9] in 2000 with a running time of at most  $\mathcal{O}^*(2^{0.30897m})$ .

**Restriction to  $k$ SAT** If we restrict ourselves to  $k$ SAT we can actually use the number of variables as a parameter and devise algorithms much better than the trivial one. The fastest known algorithm is presented in [5] and has a running time of approximately  $(2 - \frac{2}{k+1})^n$ , which for 3SAT is  $\mathcal{O}^*(1.481^n) \approx \mathcal{O}^*(2^{0.567n})$ .

This algorithm does not use branching, but is instead based on the idea of covering codes. Now if we view assignments as binary words ( $a \in \{0, 1\}^n$ ), the *Hamming distance* between two assignments is the number of positions in which they differ. The *ball* of radius  $r$  around an assignment  $a$  is the set of all assignments whose Hamming distance to  $a$  is at most  $r$ . A *code*  $C$  is a subset of possible assignments ( $C \subseteq \{0, 1\}^n$ ). Now a *covering code* of radius

$r$  is a code covering all possible assignments with balls of radius  $r$ , i.e. for all possible assignments  $a \in \{0, 1\}^n$  there exists an assignment  $a^* \in C$ , such that the Hamming distance between  $a$  and  $a^*$  is at most  $r$ .

The hard part is coming up with a good covering code, i.e. a covering code that is not too large. The article provides two different algorithms that given a radius  $r$  finds a good covering code. Then for each assignment in the code, the  $k$ SAT algorithm uses deterministic local search. Local search is simply a branching algorithm, only the depth is now bounded by  $r$ .

## 2.2 Exact Satisfiability

The following is ongoing work with Jesper Makholm Byskov and Bjarke Skjerna. We are working on exact algorithms for XSAT and X3SAT building on top of the Davis-Putnam algorithm. We simply supply new reductions rules and describe how to choose which variable to branch on, and for X3SAT we add a new trick.

For XSAT this gives us an algorithm with time complexity  $\mathcal{O}^*(2^{0.2441n})$  (branching vector  $(\mathbf{11}, \mathbf{1})$ ). This result was already published in 1980 [15] (also using the Davis-Putnam algorithm), however we hope to improve this time complexity.

For X3SAT we achieve a running time of  $\mathcal{O}^*(2^{0.1532n})$  (branching vector  $(\mathbf{10}, \mathbf{4})$ ), which improves the best known earlier result of  $\mathcal{O}^*(2^{0.18674n})$  published in [19] (also using the Davis-Putnam algorithm). I will describe a simpler version of the algorithm for X3SAT achieving a running time of  $\mathcal{O}^*(2^{0.1626n})$  (branching vector  $(\mathbf{9}, \mathbf{4})$ ) here.

Remember that in XSAT we wish to make exactly one literal true in each clause. I will use  $,$  (comma) instead of  $\vee$  (disjunction) to separate the literals in a clause in XSAT to avoid confusion.

**If Each Variable Occurs at most Twice** Before we look at the algorithm, let us note the following:

**Theorem 2.** *XSAT with each variable occurring at most twice is in  $P$ .*

*Proof.* Variables occurring once positive and once negative can be removed by resolution, i.e.  $(x, C) \wedge (\bar{x}, C') \rightsquigarrow (C, C')$ . Now make a graph with the clauses as vertices and an edge between two clauses if they share a variable. For each clause  $C_i$  containing a variable only occurring in  $C_i$  add a vertex

$D_i$  and an edge  $(C_i, D_i)$ . If the total number of vertices is odd add another vertex  $D$ . Finally connect all the  $D_i$ 's and  $D$  in a clique. See figure 2.1 for an example.

$$C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5$$

$$(y, w, u) \wedge (x, y, z) \wedge (x, v, w) \wedge (z, v, t) \wedge (t, s)$$

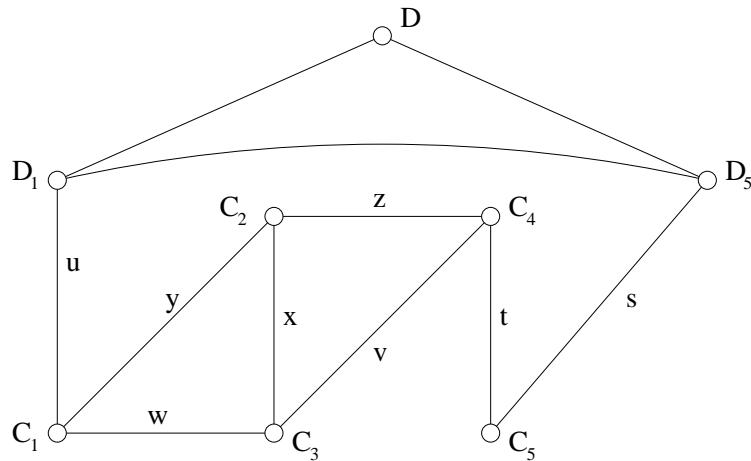


Figure 2.1: Reduction from XSAT with each variable occurring at most twice to perfect matching.

This graph has a perfect matching if and only if the formula was satisfiable in the XSAT sense, with the edges in the matching corresponding to true variables.

First assume the formula has a satisfying assignment. The edges corresponding to the true variables of this assignment will be a matching and all the  $C_i$ 's will be covered, since each clause has exactly one true variable and thus each  $C_i$  vertex is incident to exactly one edge in the matching. If any  $D_i$ 's are not covered, we can simply extend the matching, since all the  $D_i$ 's and  $D$  form a clique.

Now assume that there is a perfect matching in the graph. We simply assign true to the variables corresponding to edges in the matching. This will make exactly one variable in each clause true.

As we can determine in polynomial time if a graph has a perfect matching, XSAT in which each variable occurs at most twice is in P.  $\square$

Remember that  $x$  is an  $(a, b)$ -occurrence if  $x$  occurs  $a$  times unnegated and  $b$  times negated. The above theorem implies that we can always assume that there is some  $(a, b)$ -occurrence  $x$  having  $a + b \geq 3$ , since the problem can otherwise be solved in polynomial time.

### 2.2.1 An exact algorithm for X3SAT

**Reductions** We use the following reduction rules and always the first applicable rule from the top (this removes constants and 2-clauses first). The rules can quite easily be seen to preserve satisfiability. They also always reduce the size of the formula.

$$(true, C) \wedge F' \rightarrow F'[C := false] \quad (2.12)$$

$$(false, C) \wedge F' \rightarrow C \wedge F' \quad (2.13)$$

$$(x_i, x_j) \wedge F' \rightarrow F'[x_j := \bar{x}_i] \quad (2.14)$$

$$(x, x, y) \wedge F' \rightarrow F'[x := false, y := true] \quad (2.15)$$

$$(x, \bar{x}, y) \wedge F' \rightarrow F'[y := false] \quad (2.16)$$

$$(x_i, x_j, y) \wedge (x_i, x_j, y') \wedge F' \rightarrow (x_i, x_j, y) \wedge F'[y := y'] \quad (2.17)$$

$$(x_i, x_j, y) \wedge (x_i, \bar{x}_j, y') \wedge F' \rightarrow F'[x_i := false, y := \bar{x}_j, y' := x_j] \quad (2.18)$$

$$(x_i, x_j, y) \wedge (\bar{x}_i, \bar{x}_j, y') \wedge F' \rightarrow F'[x_j := \bar{x}_i, y := false, y' := false] \quad (2.19)$$

$$(x, C_1) \wedge \cdots \wedge (x, C_k) \wedge C \wedge F' \rightarrow C_1 \wedge \cdots \wedge C_k \wedge C \wedge F'[x := false] \quad (2.20)$$

$$C \subseteq \bigcup_{i=1}^k \{C_i\}$$

$$(x, u_1, y_1) \wedge \cdots \wedge (x, u_k, y_k) \wedge F' \rightarrow F' \quad (2.21)$$

$$x \notin F', u_i \text{ unique}$$

Note that we will not be able to use resolution in X3SAT, as this will produce 4-clauses.

**Dealing with Overlap in at most two Clauses** This is the new trick in the algorithm. Let  $Y = \{y_1, \dots, y_p\}$  be a set of variables of constant size, and  $\mathcal{C} = \{C_1, \dots, C_q\}$  the set of clauses with only variables from  $Y$ , and  $F'$  the rest of the formula. By an overlap between  $Y$  and  $F'$  in a clause, I mean that the clause  $C \in F'$  contains both a variable from  $Y$  and a variable not from  $Y$ . If there is overlap in at most two clauses, it means that  $\mathcal{C}$  is very loosely connected to the rest of the formula, and it turns out that we can then remove  $\mathcal{C}$ .

First note that if there is no overlap between  $Y$  and the rest of the formula at all, we can simply decide if  $\mathcal{C}$  is satisfiable in constant time ( $\approx 2^p$ ). If  $\mathcal{C}$

is unsatisfiable, then the whole formula is unsatisfiable. If  $\mathcal{C}$  is satisfiable, we simply remove all clauses from  $\mathcal{C}$  and continue the algorithm with the reduced formula  $F'$ .

We will let  $\overset{(-)}{x}$  denote either the literal  $x$  or the literal  $\bar{x}$ , when it is not important if the variable is negated. The  $z$  variables will here denote variables not in  $Y$ . If we have overlap in only one clause (hence only one variable), there are two possibilities. The first case looks like this:

$$\overset{(-)}{(y_i, y_j, z)}$$

In this case it is possible to remove all clauses containing a variable from  $Y$ . Either they do not restrict the value of  $z$ , and thus removing them does not influence the remaining variables. If they restrict the value of  $z$ ,  $z$  can either be substituted with a constant, or the formula is unsatisfiable. As  $Y$  is of constant size, it can in constant time be determined which is the case.

The second case looks like this:

$$\overset{(-)}{(y_i, z_1, z_2)}$$

In this case it is possible to remove all clauses containing only variables from  $Y$  (but we have to keep this one). Either they do not restrict the value of  $y_i$ , and thus removing them do not influence the remaining variables. If they restrict the value of  $y_i$ ,  $y_i$  can either be substituted with a constant, or the formula is unsatisfiable. As  $Y$  is of constant size, it can in constant time be determined which is the case.

If there is overlap in two (or more) clauses, but the overlap is only one variable, the same two cases apply. That is if we have two (or more) clauses, each containing two variables from  $Y$ , but both (all) containing the same variable  $z$  not from  $Y$ , the first case apply. If we have two (or more) clauses, each containing two variables not from  $Y$ , but only the same variable  $y_i$  from  $Y$ , the second case apply.

If there are exactly two clauses containing both variables from  $Y$  and variables which are not in  $Y$ , and the overlap is at least two variables, there are three cases:

The first case looks like this (the  $y$ 's do not have to be distinct):

$$\overset{(-)}{(y_i, y_j, z_1)} \wedge \overset{(-)}{(y_k, y_l, z_2)}$$

We define the overlap to be the variables  $z_1$  and  $z_2$ . Now it is possible to decide in constant time which truth values  $Y$  allow for  $z_1$  and  $z_2$ . If no values are allowed the formula is unsatisfiable, otherwise we remove all clauses containing a variable from  $Y$ . If all values for  $z_1$  and  $z_2$  are allowed, we do nothing more. If only one combination of truth values is allowed, we simply assign these values to  $z_1$  and  $z_2$ . If some values are allowed and some values are not, we add new clauses. The following table should be read as: If  $z_1 = v_1$  and  $z_2 = v_2$  is *not* allowed, add clause  $C$ .

$v_1$	$v_2$	$C$
<i>true</i>	<i>true</i>	$(z_1, z_2, u_1)$
<i>true</i>	<i>false</i>	$(z_1, \bar{z}_2, u_2)$
<i>false</i>	<i>true</i>	$(\bar{z}_1, z_2, u_3)$
<i>false</i>	<i>false</i>	$(\bar{z}_1, \bar{z}_2, u_4)$

The variables  $u_1, \dots, u_4$  are new variables, which we introduce. It should be noticed that if more than one of the above clauses are added, the formula can be reduced by reduction 2.17, 2.18 or 2.19. This operation removes all variables from  $Y$  without branching, and adds at most two new ones, and after reductions only one.

In the second case, we define the overlap to be  $y_i$  and  $z_l$ . The clauses look like this ( $z_l$  may be  $z_1$  or  $z_2$  or a different variable  $z_3$ , and we may have  $y_i = y_j$  or  $y_i = y_k$ ):

$$\binom{(-)}{y_i}, \binom{(-)}{z_1}, \binom{(-)}{z_2} \wedge \binom{(-)}{y_j}, \binom{(-)}{y_k}, \binom{(-)}{z_l}$$

Here we can do the same trick. First decide the allowed truth value combinations for  $y_i$  and  $z_l$ . If none are allowed, the formula is unsatisfiable. Otherwise remove all clauses containing a variable from  $Y$ , except for the clause  $\binom{(-)}{y_i}, \binom{(-)}{z_1}, \binom{(-)}{z_2}$ . Once again if all values are allowed, we do nothing more. If only one combination of truth values is allowed, we simply assign these values to  $y_i$  and  $z_l$ . If some values are allowed and some values are not, we add new clauses according to the table above, only with  $z_1$  and  $z_2$  replaced by  $y_i$  and  $z_l$ . This operation removes all variables from  $Y$  except  $y_i$ , and adds at most one new.

In the third case, we define the overlap to be  $y_i$  and  $y_j$ . The clauses look like this (the  $z$ 's do not have to be distinct):

$$\binom{(-)}{y_i}, \binom{(-)}{z_1}, \binom{(-)}{z_2} \wedge \binom{(-)}{y_j}, \binom{(-)}{z_k}, \binom{(-)}{z_l}$$

Again we can do the same trick, by deciding the allowed truth values for  $y_i$  and  $y_j$ . If none, the formula is unsatisfiable. If any, remove all clauses containing only variables from  $Y$  but keep these two clauses. If only one combination of truth values is allowed, assign these values to  $y_i$  and  $y_j$ . If some values are allowed and some values are not, add new clauses according to the table above, only with  $z_1$  and  $z_2$  replaced by  $y_i$  and  $y_j$ . This operation removes all variables from  $Y$  except  $y_i$  and  $y_j$ , and adds at most one new.

This should enable us to deal with all possible overlaps in at most two clauses, and we will use this trick in the algorithm.

Finally we will need this observation: If there is overlap in more than two clauses, but the overlap is only in two variables  $z_1$  and  $z_2$  not in  $Y$ , we can still use the trick from the first case for two clauses.

**Theorem 3.** *The Davis-Putnam algorithm with the above reduction rules and the new trick for dealing with overlap in at most two clauses solves X3SAT in time  $\mathcal{O}^*(2^{0.1532n})$ .*

*Proof.* Look at an  $(a,b)$ -occurrence  $x$  with  $a + b \geq 3$ . Setting  $x$  to *true* removes at least  $2a + b + 1$  variables. The  $2a$  variables come from the  $a$  clauses where  $x$  occurs unnegated, since no clauses have more than one variable in common (reductions 2.17, 2.18 and 2.19), and setting  $x$  to *true* forces all the other variables in these clauses to be false. The  $b$  variables come from the  $b$  clauses where  $x$  occurs negated, as 2-clauses can be removed by reduction 2.14. Finally the 1 is  $x$  itself. Setting the variable to *false* then removes at least  $2b + a + 1$  variables by a similar argument. For  $a + b \geq 4$  this gives in the worst case a branching vector of **(9, 5)**. Remember we are aiming for a worst case branching vector of **(9, 4)**.

Now let us look at the hard cases:  $(3, 0)$ -occurrences and  $(2, 1)$ -occurrences. So far the above gives us branching vector **(7, 4)** for  $(3, 0)$ -occurrences and **(6, 5)** for  $(2, 1)$ -occurrences. To improve this we need to look at the overlap with the rest of the formula.

**(3, 0)-occurrences** By use of reductions we can assume that the clauses in which the  $(3, 0)$ -occurrence,  $x$ , occurs look as follows:

$$(x, \overset{(-)}{y_1}, \overset{(-)}{y_2}) \wedge (x, \overset{(-)}{y_3}, \overset{(-)}{y_4}) \wedge (x, \overset{(-)}{y_5}, \overset{(-)}{y_6})$$

Let  $Y$  be the set  $\{y_1, \dots, y_6\}$ . We will let  $y$ -variables be variables in  $Y$ ,  $z$ -variables be variables not in  $Y$  (and not  $x$ ) and  $w$ -variables be arbitrary

variables (also not  $x$ ).

If there is overlap between  $Y \cup x$  and the rest of the variables in at most two clauses or if there is overlap in at most two  $z$ -variables, we apply the above trick, where  $p = 7$  and  $q = 3$ . Otherwise at least three other clauses contains both a variable from  $Y$  and a variable not in  $Y$ , and there are at least three different variables not from  $Y$  in these clauses.

$$\binom{(-)}{y_i}, \binom{(-)}{w_1}, \binom{(-)}{z_1} \wedge \binom{(-)}{y_j}, \binom{(-)}{w_2}, \binom{(-)}{z_2} \wedge \binom{(-)}{y_k}, \binom{(-)}{w_3}, \binom{(-)}{z_3}$$

When setting  $x$  to true, we remove at least two extra variables: If some of the  $w$ 's are  $z$ 's, some of the  $z$ 's may be the same. But remember we have three distinct  $z$ 's, thus when we remove at least one  $z$ -variable in each clause (by reduction rules 2.12, 2.13 and 2.14), we must have removed two distinct  $z$ 's. If all the  $w$ 's are  $y$ 's, the  $z$ 's are distinct, and we remove all three.

Thus we remove at least two extra variables in the branch, where we set  $x$  to true, and we get a branching vector of at least  $(9, 4)$ .

**(2,1)-occurrences** By use of reductions we can assume that the clauses in which the (2,1)-occurrence,  $x$ , occurs look as follows:

$$\binom{(-)}{x}, \binom{(-)}{y_1}, \binom{(-)}{y_2} \wedge \binom{(-)}{x}, \binom{(-)}{y_3}, \binom{(-)}{y_4} \wedge \binom{(-)}{\bar{x}}, \binom{(-)}{y_5}, \binom{(-)}{y_6}$$

Again if there is overlap between  $Y \cup x$  and the rest of the variables in at most two clauses or if there is overlap in at most two  $z$ -variables, we apply the above trick, where  $p = 7$  and  $q = 3$ . Otherwise at least three other clauses contains both a variable from  $Y$  and a variable not in  $Y$ , and there are at least three different variables not from  $Y$  in these clauses.

$$\binom{(-)}{y_i}, \binom{(-)}{w_1}, \binom{(-)}{z_1} \wedge \binom{(-)}{y_j}, \binom{(-)}{w_2}, \binom{(-)}{z_2} \wedge \binom{(-)}{y_k}, \binom{(-)}{w_3}, \binom{(-)}{z_3}$$

Now we branch on setting  $x$  either true or false, and in one of these branches we will set  $y_i$  to false, in one we will set  $y_j$  to false and in one we will set  $y_k$  to false. By the same argument as in the case before, if we look at both branches, we will have removed at least one  $z$  from each clause, and at least two of them are distinct, thus we get branching vector  $(7, 6)$  or  $(8, 5)$ , of which the worst is  $(8, 5)$ .

**Final Result** Thus the worst case branching vector is  $(\mathbf{9}, \mathbf{4})$ , which gives us the time complexity  $\mathcal{O}^*(2^{0.1626n})$ .  $\square$

# Chapter 3

## Graph Theoretical Problems

In an undirected graph  $G = (V, E)$ , we will let  $n$  denote the number of vertices and  $m$  the number of edges.

The neighbours of vertex  $v \in V$  is the set  $N(v) = \{u \in V | (v, u) \in E\}$ . The number of neighbours of a vertex  $v \in V$  is called the degree of  $v$  and is denoted  $d(v) = |N(v)|$ .

An independent set in a graph  $G$  is a subset  $I \subseteq V$  such that no two vertices in  $I$  has an edge between them, i.e.  $\forall u, v \in I : (u, v) \notin E$ .

A bipartite graph is a graph with the property, that you can partition  $V$  into  $L$  and  $R$  ( $L \cup R = V$ ,  $L \cap R = \emptyset$ ), such that  $L$  and  $R$  are both independent sets.

The subgraph induced by  $S \subseteq V$  is defined as  $G[S] = (S, E')$ , where  $E' = \{(u, v) \in E | u, v \in S\}$ . A bipartite (induced) subgraph is simply an induced subgraph which is bipartite.

A set  $S$  (or subgraph  $G[S]$ ) with property  $P$  is *maximal*, if adding any vertex  $v \in V \setminus S$  to  $S$  will violate the property.

A  $k$ -colourable graph is a graph  $G$  with the property that you can colour all the vertices of  $G$  with only  $k$  different colours, such that no neighbours have the same colour. In a  $k$ -colouring of a graph, each colour  $i$  defines a subset  $V_i \subseteq V$  of all the vertices with colour  $i$ . Each  $V_i$  is an independent set. Also note that a 2-colourable graph is the same as a bipartite graph. The minimum number  $k$ , for which a graph is  $k$ -colourable is called the chromatic number,  $\chi(G)$ .

## 3.1 Maximal Independent Sets

I will describe an algorithm enumerating all maximal independent sets in a graph. As we will see, there can be exponentially many maximal independent sets in a graph, thus we cannot hope to list them all in less than exponential time or space. The motivation for finding an enumeration algorithm for maximal independent sets, is that it can be used in algorithms for the NP-complete problems MAXIMUM INDEPENDENT SET and  $k$ -COLOURING.

### 3.1.1 An Enumeration Algorithm

First let us look at some results on maximal independent sets. Already in 1965, Moon and Moser showed this theorem [16]:

**Theorem 4.** *The number of maximal independent sets in a graph of size  $n$  is at most  $3^{n/3}$ .*

They also showed that there exist graphs with this many maximal independent sets, namely the graphs consisting of disjoint 3-cliques. This tells us that an enumeration algorithm running in time  $\mathcal{O}^*(3^{n/3})$  is optimal<sup>1</sup>, and we will see that the following algorithm inspired by [7] actually runs in time  $\mathcal{O}^*(3^{n/3}) \approx \mathcal{O}^*(2^{0.5283n})$ .

```

MIS( $S, I$ )
  if  $S = \emptyset$  then
    if  $I$  maximal2 then output  $I$  fi
  fi
  if  $\exists v \in S : d(v) \geq 3$  then
    MIS( $S \setminus (\{v\} \cup N(v)), I \cup \{v\}$ );
    MIS( $S \setminus \{v\}, I$ );
  else let  $v$  be a vertex with minimal degree in  $S$ ;
    MIS( $S \setminus (\{v\} \cup N(v)), I \cup \{v\}$ );
    for all  $u \in N(v)$ :
      MIS( $S \setminus (\{u\} \cup N(u)), I \cup \{u\}$ );
    end for
  fi

```

---

<sup>1</sup>Here we of course mean optimal in the worst-case sense. An algorithm with only polynomial delay between the output of produced sets can be found in [10].

<sup>2</sup>As we do not force a neighbour to be in  $I$  in the case where  $d(v) \geq 3$  (the call MIS( $S \setminus \{v\}, I$ )),  $I$  might not be maximal.

**Theorem 5.** *The MIS algorithm has time (and space) complexity  $\mathcal{O}^*(3^{n/3})$ .*

*Proof.* For the case  $d(v) \geq 3$  we get a branching vector (with respect to the number of vertices  $n$ ) of at least  $(4, 1)$ . For the case  $d(v) = 0$  we only make one recursive call, for the case  $d(v) = 1$  we get the branching vector  $(2, 2)$  and for the case  $d(v) = 2$  we get a branching vector of  $(3, 3, 3)$ . The worst of these is  $(3, 3, 3)$ , which gives us the running time of  $\mathcal{O}^*(3^{n/3}) \approx \mathcal{O}^*(2^{0.5283n})$ . The number of maximal independent sets output by the algorithm is therefore also  $\mathcal{O}^*(3^{n/3})$ .  $\square$

For maximal independent sets of smaller size better bounds are known (I will not prove these results), and also enumeration algorithms much like this one exist for maximal independent sets of size at most  $k$ .

The first result is shown by Eppstein in a paper from 2001 [7]:

**Theorem 6.** *The number of maximal independent sets of size at most  $k$  in a graph of size  $n$  is at most  $3^{4k-n}4^{n-3k}$ , and there is an algorithm for listing all maximal independent sets smaller than  $k$  in an  $n$ -vertex graph in time  $\mathcal{O}^*(3^{4k-n}4^{n-3k})$ .*

This result is tight for  $n/4 \leq k \leq n/3$ . For  $k > n/3$  the best bound is given by Theorem 4. For  $k < n/4$  a better bound is given in [18] (for  $n/4 \leq k \leq n/3$  the result is the same as the one above):

**Theorem 7.** *The number of maximal independent sets of size at most  $k$ , where  $1 \leq k \leq n/3$ , in a graph of size  $n$  is at most*

$$\lfloor n/k \rfloor^{k-(n \bmod k)} (\lfloor n/k \rfloor + 1)^{n \bmod k}.$$

Moreover, there is an algorithm listing all maximal independent sets of size at most  $k$  in time  $\mathcal{O}^*\left(\lfloor n/k \rfloor^{k-(n \bmod k)} (\lfloor n/k \rfloor + 1)^{n \bmod k}\right)$ .

### 3.1.2 Applications

The obvious NP-complete problem for which we can use the enumeration algorithm is the MAXIMUM INDEPENDENT SET problem.

**Definition 2.** *The MAXIMUM INDEPENDENT SET problem is: Given a graph  $G = (V, E)$ , find the largest independent set in this graph.*

The MAXIMUM INDEPENDENT SET problem was first shown NP-complete using a reduction from SAT by Karp in 1972 [11].

The naive algorithm of testing all subsets gives us an  $\mathcal{O}^*(2^n)$  algorithm, but of course a maximum independent set of a graph is also a maximal independent set. Thus we can use the enumeration algorithm for maximal independent sets and simply only remember the largest set at any time, giving us an algorithm running in time  $\mathcal{O}^*(3^{n/3}) \approx \mathcal{O}^*(2^{0.5283n})$  and polynomial space.

This, however, is far from the best. Already in 1977 Tarjan and Trojanowski presented an algorithm running in time  $\mathcal{O}^*(2^{n/3})$  [22]. The algorithm as well as the time analysis is also based on case analysis (branching). This algorithm has been improved a number of times and using computer generated cases Robson managed to obtain a running time of  $\mathcal{O}^*(2^{0.251})$  [20].

Another problem for which we are able to use the enumeration algorithm is the  $k$ -COLOURING problem.

**Definition 3.** *The  $k$ -COLOURING problem is: Given graph  $G = (V, E)$ , is it  $k$ -colourable?*

This problem was first shown to be NP-complete for  $k \geq 3$  by Karp in 1972 [11]. Notice that for  $k = 1$ , it simply corresponds to checking if there are any edges at all, and for  $k = 2$  it corresponds to checking if the graph is bipartite, which we know can be done in polynomial time by the greedy colouring algorithm.

For  $k \geq 3$ , first note that simply trying all possible colourings of vertices and then checking if they are valid (no edge connects two vertices with the same colour) will take time  $\mathcal{O}^*(k^n)$ , as each vertex can have  $k$  different colours and checking if the colouring is valid can be done in time proportional to the number of edges.

This means that the problem of 3-colouring can be solved in time  $\mathcal{O}^*(3^n)$ , which is not very impressive. However if we have a  $k$ -colouring of a graph, then the subgraph induced by one of the colours can be extended to a maximal independent set of size at least  $n/k$ .

We can simply pick the 'largest colour' (the set defined by the most used colour) and extend it if possible. The 'colour' was already independent and of size at least  $n/k$  and now it is also maximal.

Thus for each maximal independent set, we can simply test whether the remaining graph is bipartite as noted by Lawler [13]. This gives us an algo-

rithm running in time proportional to finding all maximal independent sets as checking if a graph is bipartite is polynomial, that is  $\mathcal{O}^*(3^{n/3}) \approx \mathcal{O}^*(2^{0.5283n})$ .

The best known result for 3-colouring however appears in [2], where Beigel and Eppstein present an algorithm solving 3-colouring in time  $\mathcal{O}^*(1.3289^n) \approx \mathcal{O}^*(2^{0.4102n})$ . This algorithm is basically a branching algorithm, where we branch on restrictions of allowed colours for a vertex, on top of which a number of reduction rules are added.

It turns out that combining the approaches yields the best known time complexity for 4-colouring [18]. Again we use that if we have a  $k$ -colouring of a graph, the subgraph induced by one of the colours can be extended to a maximal independent set of size at least  $n/k$ . We simply find all maximal independent sets of size at least  $n/4$ , and for each of these test if the remaining graph is 3-colourable using the best known algorithm for 3-colouring. This yields a time complexity of  $\mathcal{O}^*(1.7504^n) = \mathcal{O}^*(2^{0.8077n})$ .

The same approach works for 5-colouring, and adding some extra tricks also for 6- and 7-colouring, but for higher  $k$ , the best known time complexity is  $\mathcal{O}^*((4/3 + 3^{4/3}/4)^n) = \mathcal{O}^*(2.4150^n)$  achieved by the chromatic number algorithm by Eppstein [7].

## 3.2 Maximal Bipartite Subgraphs

Using the enumeration algorithm for maximal independent sets, we can construct an algorithm enumerating all maximal bipartite subgraphs. As with the algorithm for maximal independent sets, we hope to be able to use a good enumeration algorithm for maximal bipartite subgraphs to devise a good algorithm for  $k$ -COLOURING. The algorithm can of course also be used for MAXIMUM BIPARTITE SUBGRAPH.

### 3.2.1 An Enumeration Algorithm

The following bounds on the number of maximal bipartite subgraphs in a graph is joint work with Jesper Makholm Byskov and Bjarke Skjernaa from [14].

**Theorem 8.** *Any graph contains at most  $\mathcal{O}(12^{n/4}) = \mathcal{O}(1.8613^n)$  maximal bipartite subgraphs. Moreover, there is an algorithm that takes as input a graph and outputs all its maximal bipartite subgraphs in time  $\mathcal{O}^*(1.8613^n)$ .*

*Proof.* First we note that a maximal bipartite subgraph of a given graph  $G$  consists of a maximal independent set  $M_1$  of  $G$  and a maximal independent set  $M_2$  of the remaining graph  $G[V \setminus M_1]$  with  $|M_1| \geq |M_2|$ . To see this, take the largest colour in the maximal bipartite subgraph (or 2-colourable subgraph) and extend it to a maximal independent set. As we have argued before this is possible. Now the other colour must be a maximal independent set of the remaining graph, since it was already independent and if it is not maximal in the remaining graph, we can add a vertex to our maximal bipartite subgraph, but then it was not maximal to begin with.

Now let  $G$  be the given graph. To find all maximal bipartite subgraphs of  $G$ , our algorithm generates all maximal independent sets of  $G$  and for each generates all no larger maximal independent sets of the remaining graph. If their union is a maximal bipartite subgraph, the algorithm outputs it.<sup>3</sup> Let  $I_k(G)$  denote the set of all maximal independent sets of size at most  $k$  in  $G$ . Then the number of maximal bipartite subgraphs of  $G$  is at most

$$\sum_{k=1}^n \sum_{\substack{I \in I_k(G) \\ |I|=k}} |I_k(G[V \setminus I])| \leq \sum_{k=1}^n |I_k(G)| \cdot \max_{\substack{I \subseteq V \\ |I|=k}} |I_k(G[V \setminus I])|.$$

We split the sum in two and use the following bounds on the number of maximal independent sets in a graph. Eppstein shows that  $|I_k(G)| \leq 3^{4k-n} 4^{n-3k}$  (see Theorem 6), and Moon and Moser show that any graph can have at most  $3^{n/3}$  maximal independent sets in total (see Theorem 4). Using these two bounds we get that the sum is at most

$$\sum_{k=1}^{\lfloor \frac{n}{4} \rfloor} 3^{4k-n} 4^{n-3k} 3^{4k-(n-k)} 4^{(n-k)-3k} + \sum_{k=\lfloor \frac{n}{4} \rfloor + 1}^n 3^{4k-n} 4^{n-3k} 3^{(n-k)/3}.$$

Moving the terms not depending on  $k$  outside the sums and using the fact that the sums are geometric series we get that the whole expression is  $\mathcal{O}(12^{n/4}) = \mathcal{O}(1.8613^n)$ .

Our algorithm generates all maximal independent sets of the graph in time  $\mathcal{O}^*(3^{n/3})$ . For those of size  $k \leq n/4$  we use the algorithm of Eppstein (Theorem 6) to generate the maximal independent sets of size at most  $k$  of

---

<sup>3</sup>The union might not be a maximal bipartite subgraph: In the 6-cycle for example, two opposite vertices form a maximal independent set, but their union with a maximal independent set of the remaining graph does not form a maximal bipartite subgraph.

the remaining graphs, and for those of size  $k > n/4$  we generate all maximal independent sets of the remaining graph in time  $\mathcal{O}^*(3^{k/3})$ . The algorithm then runs in time  $\mathcal{O}^*(1.8613^n) \approx \mathcal{O}^*(2^{0.8963n})$ .  $\square$

When we looked at the enumeration algorithm for maximal independent sets, we noted that there actually exists graphs with as many maximal independent sets as the upper bound, thus we couldn't hope to do better. This is not the case for maximal bipartite subgraphs. The best lower bound on the maximum number of maximal independent sets is the following (also from [14]):

**Theorem 9.** *There exists an infinite family of graphs all having  $105^{n/10} \approx 1.5926^n$  maximal bipartite subgraphs.*

*Proof.* Look at the graph in Figure 3.1, and let a *pair* denote a vertex on the outer 5-cycle and the nearest vertex on the inner 5-cycle.

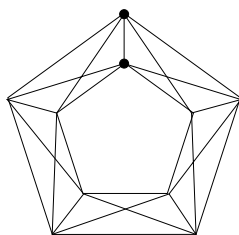


Figure 3.1: Generating graph with a pair marked.

The graph has  $5 \cdot 2^4 = 80$  maximal bipartite subgraphs containing one vertex from four of the pairs (see Figure 3.2(a)),  $5 \cdot 2^2 = 20$  containing one pair and one vertex from each of the opposite pairs (see Figure 3.2(b)) and five containing two pairs (see Figure 3.2(c)). Thus it has 105 maximal bipartite subgraphs.

The infinite family consists of disconnected copies of this graph, the  $k$ 'th one having  $k$  copies. The maximal bipartite subgraphs of a disconnected graph are exactly the union of one maximal bipartite subgraph of each connected component. Their number thus equals the product of the number of maximal bipartite subgraphs of each component. This gives us a lower bound of  $105^{n/10} \approx 1.5926^n$ .  $\square$

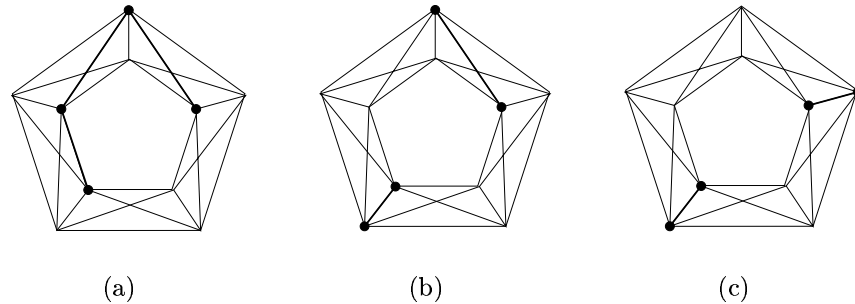


Figure 3.2: The three different types of maximal bipartite subgraphs.

### 3.2.2 Applications

Our enumeration algorithm for maximal bipartite subgraphs, can be used for the following problem:

**Definition 4.** *The MAXIMUM BIPARTITE SUBGRAPH problem is: Given a graph  $G = (V, E)$  and a positive integer  $K$ , is there a subset  $V' \subseteq V$  with  $|V'| \geq K$  such that the subgraph induced by  $V'$  is bipartite?*

This is of course the decision version. In the maximisation version we have no  $K$  as input, but simply wish to find the highest  $K$  for which we can answer yes.

The problem is actually (like MAXIMUM INDEPENDENT SET) a special case of the INDUCED SUBGRAPH WITH PROPERTY  $\Pi$  problem [8], which was proved NP-complete<sup>4</sup> by Mihalis Yannakakis [23] in 1978 in a somewhat complicated proof. However, when we know that the property in question is if the graph is bipartite, there is a simple proof. As this is not a very well-known problem, neither is the proof for NP-completeness, so I have chosen to include it here.

**Theorem 10.** MAXIMUM BIPARTITE SUBGRAPH is NP-complete.

*Proof.* The problem is easily seen to be in NP. To show NP-hardness we describe a simple reduction from NAE3SAT, which was one of the problems

---

<sup>4</sup>Provided that property  $\Pi$  holds for arbitrarily large graphs, does not hold for all graphs, is “hereditary”, i.e. holds for all induced subgraphs of  $G$  whenever it holds for  $G$ , and one can determine in polynomial time whether  $\Pi$  holds for a given graph.

proven NP-complete by Schaefer [21]. The reduction from NAE3SAT to MAXIMUM BIPARTITE SUBGRAPH uses a simple triangle gadget.

Formally, we are given an instance  $F$  of NAE3SAT with  $m$  clauses  $C_1, \dots, C_m$ , and each clause has exactly three literals  $C_i = (\alpha_{i1}, \alpha_{i2}, \alpha_{i3})$ . Our reduction  $R(F) = (G, K)$  has goal  $K = 2m$ , and  $G = (V, E)$  is the following graph:  $V = \{v_{ij} : i = 1, \dots, m; j = 1, 2, 3\}$ ;  $E = \{(v_{ij}, v_{ik}) : i = 1, \dots, m; j \neq k\} \cup \{(v_{ij}, v_{lk}) : i \neq l, \alpha_{ij} = \neg\alpha_{lk}\}$ . See figure 3.3 for an example.

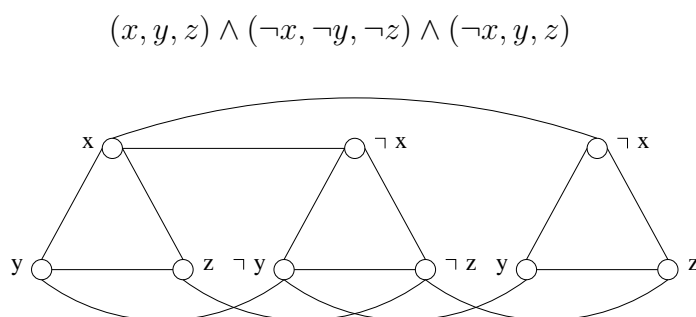


Figure 3.3: Reduction from NAE3SAT to MAXIMUM BIPARTITE SUBGRAPH.

We claim that a bipartite subgraph of size  $K$  exists in  $G$  if and only if  $F$  has a truth assignment that makes at least one literal true and at least one literal false in each clause.

Suppose that a bipartite subgraph of size  $K = 2m$  exists in  $G$ . Then each triangle must contribute two vertices, and if we two-colour the subgraph two vertices from the same triangle must receive different colours. Choose one colour to be true and the other false (vertices/variables without a colour will either be determined by their neighbours or we can choose). This assignment will assure at least one true literal and at least one false literal in each clause. Note that the edges between vertices representing a certain variable and the vertices representing the negation of this variable assures the opposite truth values for a variable and its negation, so we have a proper assignment.

Conversely if an assignment exists that makes at least one literal true and at least one literal false in each clause of  $F$ , we can pick a vertex corresponding to a true literal and a vertex corresponding to a false literal from each triangle. This set will have size  $K = 2m$  and it is bipartite, because simply using the assignment of true and false to the literals we get a two-colouring of the

vertices: If we look at a vertex with the colour true, it has two neighbours in the triangle. The one we picked has the colour false and one wasn't picked. The rest of the neighbours all represent the negation of this literal, and hence are either false or not picked. □

The obvious algorithm for solving MAXIMUM BIPARTITE SUBGRAPH tests all induced subgraphs to see if they are bipartite and larger than the previous ones, which gives us a running time of  $\mathcal{O}^*(2^n)$ . A better algorithm checks only all the maximal bipartite subgraphs. Thus using our enumeration algorithm for maximal bipartite subgraphs remembering only the largest maximal bipartite subgraph so far, we obtain an algorithm only using time  $\mathcal{O}^*(1.8613^n) \approx \mathcal{O}^*(2^{0.8963n})$  and polynomial space. As this is not a very well known problem, I have not found any better exact algorithms for MAXIMUM BIPARTITE SUBGRAPH.

As our bound on maximal bipartite subgraphs is based on the bound for maximal independent sets, we cannot use this to devise improved algorithms for colouring. However we have reason to believe that the actual upper bound on maximal bipartite subgraphs is lower. Our algorithm may find two independent sets, only to discover, that while combining them of course yields a bipartite subgraph, it might not be maximal.

The worst case graphs in the proof of our upper bound are those having  $4^{n/4}$  maximal independent sets of size  $\frac{n}{4}$ . The only graphs achieving this are graphs that consist of a union of disconnected 4-cliques (by [7]), and these have only  $6^{n/4} \approx 1.5651^n$  maximal bipartite subgraphs. Also, if we use the same approach as in the proof of Theorem 8 to prove an upper bound on the number of maximal 3-colourable subgraphs of a graph we get a bound of  $\mathcal{O}(2.2680^n)$ , which is clearly too high.

Thus if we were able to find a better enumeration algorithm for maximal bipartite subgraphs, this could possibly be used in better colouring algorithms.

# Chapter 4

## Future Work

In part A of my PhD studies I have concentrated mostly on exact algorithms, and I would like to continue along these lines in part B.

**Exact Satisfiability** At the moment Jesper Makholm Byskov, Bjarke Skjernaa and I are writing a paper improving the best known time complexity for solving X3SAT (see Section 2.2) and hopefully also XSAT. Our algorithm for X3SAT has time complexity  $\mathcal{O}^*(2^{0.1532n})$  with the worst case branching vector being  $(10, 4)$ , which is the best known time complexity for X3SAT, and we hope to be able to improve this by a more careful analysis, so we get a branching vector of  $(11, 4)$ .

The algorithm for XSAT uses the same approach as the X3SAT algorithm, only with some additional reduction rules and resolution. We hope to be able to improve the time complexity for the XSAT algorithm by considering how to deal with clauses sharing more than one variable. We have reduction rules, that take care of clauses sharing all or all but one variables. When fewer variables are shared, i.e. we have  $(C, C_1) \wedge (C, C_2)$ , where  $|C|, |C_1|, |C_2| \geq 2$ , it seems promising to branch on the truth value of  $C$ , instead of just a single variable.

**More Exact Algorithms** The algorithms for XSAT and X3SAT are traditional branching algorithms. While it is probably possible to improve the time complexities for some NP-complete problems, simply by refining known branching algorithms, it seems like new approaches may give us a better understanding of the field.

It could for instance be interesting to see if the trick introduced in the X3SAT algorithm to deal with parts of a formula only loosely connected to the main formula, could also be used in algorithms for other satisfiability problems

**Parameterised Algorithms** The area of parameterised complexity offers a different perspective on exact algorithms. The idea in parameterised algorithms for NP-complete problems is to try and shift the exponential behaviour from the traditional input parameter  $n$  (the length of (part of) the input) to a different and hopefully much smaller parameter.

A parameterised problem is a problem, where we are given an integer parameter  $0 < k \leq n$  as part of the input. Look for example at the following problem:

**Definition 5.** *The MINIMUM VERTEX COVER problem is: Given a graph  $G = (V, E)$  and a nonnegative integer  $k$ , is there a subset of vertices  $C \subseteq V$  with  $k$  or fewer vertices such that each edge in  $E$  has at least one of its endpoints in  $C$ ?*

A parameterised problem is *fixed-parameter tractable* if there is an algorithm solving the problem with running time  $f(k) \cdot n^{\mathcal{O}(1)}$ , where  $f$  is an arbitrary function on nonnegative integers. The corresponding complexity class is called FPT.

MINIMUM VERTEX COVER is NP-complete (the complement is a maximum independent set), but it is also fixed-parameter tractable. Consider the following trivial fixed-parameter algorithm: As long as there are uncovered edges, pick an arbitrary uncovered edge  $(u, v)$  and branch into two cases:

1.  $u$  is in the vertex cover, or
2.  $v$  is in the vertex cover.

Repeat this until all edges are covered or more than  $k$  vertices had to be taken into the vertex cover. This yields a bounded search tree of size  $2^k$  and the corresponding algorithm has worst case time complexity  $\mathcal{O}(2^k \cdot n^2)$ . This can of course be improved, and actually the best known parameterised algorithm for MINIMUM VERTEX COVER has time complexity  $\mathcal{O}(kn + 1.271^k)$  [3, 17].

However not all problems are fixed-parameter tractable. Like the polynomial hierarchy, a hierarchy for parameterised complexity has been introduced,

see for example [1]. The class FPT is a subclass of the class of parameterised intractable problems  $W[1]$ , and if a  $W[1]$ -complete problem can be proven to belong to FPT, the hierarchy will collapse. MAXIMUM INDEPENDENT SET is one of the problems, which have been proven  $W[1]$ -complete, thus we do not expect MAXIMUM INDEPENDENT SET to be in FPT.

It could be interesting to see if more of the techniques known from the area of exact algorithms can be applied to parameterised algorithm, thus giving us better algorithms. Also since the area is quite new, there are still problems, for which we do not know if they belong to FPT or are hard for the class of parameterised intractable problems. It would be nice to be able to classify more problems in the parameterised complexity hierarchy.



# Acknowledgements

I am very grateful to both Jesper Makhholm Byskov and Bjarke Skjerna. Working with Jesper and Bjarke over the last two years has been inspiring – and fun – and helped keeping me on track.

I would also like to thank Sven Skyum, who is Bjarke's and my supervisor, and Peter Bro Miltersen, who is Jesper's supervisor, for their help and contributions to our work along the way.

And thank you to Lone Asferg Laursen, Kåre Fiedler Christiansen, Mads Jurik, and once again Jesper, Bjarke and Sven for helpful suggestions, comments and corrections to this progress report. Also thanks to all the other people, who offered to help.



# Bibliography

- [1] J. Alber, J. Gramm, and R. Niedermeier. Faster exact algorithms for hard problems: A parameterized point of view. *Discrete Mathematics*, 229(1):3–27, 2001.
- [2] Richard Beigel and David Eppstein. 3-coloring in time  $\mathcal{O}(1.3289^n)$ . ACM Computing Research Repository, June 2000.
- [3] Jianer Chen, Iyad Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. In *Workshop on Graph-Theoretic Concepts in Computer Science*, pages 313–324, 1999.
- [4] S. A. Cook. The complexity of theorem-proving procedures. In *proc. 3rd ann. ACM symp. on theory of computing, association for computing machinery, NY*, pages 151–158, 1971.
- [5] Evgeny Dantsin, Andreas Goerdt, Edward A. Hirsch, Ravi Kannan, Jon Kleinberg, Christos Papadimitriou, Prabhakar Raghavan, and Uwe Schöning. A deterministic  $(2 - 2/(k + 1))^n$  algorithm for k-SAT based on local search, 2000.
- [6] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.
- [7] David Eppstein. Small maximal independent sets and faster exact graph coloring. In *Proc. 7th Worksh. Algorithms and Data Structures*, volume 2125 of *Lecture Notes in Computer Science*, pages 462–470. Springer-Verlag, August 2001.
- [8] M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

- [9] Edward A. Hirsch. New worst-case upper bounds for SAT. *Journal of Automated Reasoning*, 24(4):397–420, 2000.
- [10] David S. Johnson, Mihalis Yannakakis, and Christos H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.
- [11] R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.
- [12] O. Kullmann and H. Luckhardt. Deciding propositional tautologies: Algorithms and their complexity, 1997.
- [13] Eugene L. Lawler. A note on the complexity of the chromatic number problem. *Information Processing Letters*, 5(3):66–67, August 1976.
- [14] Bolette Ammitzbøll Madsen, Jesper Makholm Nielsen, and Bjarke Skjernaa. On the number of maximal bipartite subgraphs of a graph. Report Series RS-02-17, BRICS, Department of Computer Science, Aarhus University, April 2002.
- [15] Burkhard Monien, Ewald Speckenmeyer, and Oliver Vornberger. Upper bounds for covering problems. Bericht 7, Universität Paderborn, 1980.
- [16] J. W. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3:23–28, 1965.
- [17] Rolf Niedermeier and Peter Rossmanith. On efficient fixed parameter algorithms for weighted vertex cover. In *ISAAC*, pages 180–191, December 2000.
- [18] Jesper Makholm Nielsen. On the number of maximal independent sets in a graph. Report Series RS-02-15, BRICS, Department of Computer Science, Aarhus University, April 2002.
- [19] Stefan Porschen, Bert Randerath, and Ewald Speckenmeyer. X3SAT is decidable in time  $2^{n/5}$ , 2002.
- [20] J. M. Robson. Finding a maximum independent set in time  $\mathcal{O}(2^{n/4})$ ? Manuscript, January 2001.

- [21] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226, 1978.
- [22] Robert Endre Tarjan and Anthony E. Trojanowski. Finding a maximum independent set. *SIAM Journal on Computing*, 6(3):537–546, September 1977.
- [23] M. Yannakakis. Node- and edge-deletion NP-complete problems, 1978.