Diese Arbeit wurde vorgelegt am LuFG Theorie hybrider Systeme

BACHELORARBEIT

HEURISTICAL VARIABLE ORDERING IN THE CYLINDRICAL ALGEBRAIC DECOMPOSITION

Kristian Covic

Prüfer:
Prof. Dr. Erika Ábrahám
Prof. Thomas Noll

Zusätzlicher Berater:
Jasper Nalbach

Aachen, 16.01.2023

Abstract

Satisfiability Modulo Theories (SMT) solving using the Cylindrical Algebraic Decomposition (CAD) method provides a complete method to decide satisfiability in quantifier-free non-linear real arithmetic (QF_NRA) and is therefore an active topic of research. The runtime of the CAD procedure is bounded by a double exponential term in the number of variables, which has motivated various optimization techniques to improve performance in practical problems. The order of variables can have a significant impact on the runtime, which has lead to various heuristics for finding a good variable ordering for a given input problem. In this work, existing heuristics are presented and evaluated. Based on the results, new variants of a chordality-based heuristic are proposed, which aim to include more information on the polynomial set to arrive at a more optimal variable ordering.

Contents

Chapter 1

Introduction

SMT solving is been an active topic of research for a long time, as an efficient approach to decide satisfiability for first-order logic over given theories. In this framework, a solver for the Boolean Satisfiability Problem (SAT) is combined with a theory solver. The SAT component only considers the logical structure of the formula, finding sets of theory constraints that satify the full formula. When such a set is found, it is passed to the theory solver to decide its satisfiability. Depending on the answer, the SMT solver can conclude satisfiability for the full formula or continue to explore the boolean search space. The choice of the theory module depends on the logic to solve. For QF_NRA, the CAD algorithm can be employed, which is a complete procedure for quantifier elimination in non-linear real arithmetic (NRA). While CAD is one of the few complete methods for this task, its runtime is bounded by a double exponential term. Thus, it is often infeasible to use as a standalone theory solver when quantifier elimination is not needed. In a SMT solver, the CAD will only be computed on partial problems, which can offer a better average-case performance in practical applications.

CAD receives a set of multivariate polynomial constraints as an input and performs a two-step process to decide satisfiability: In the *projection phase*, a projection operator is repeatedly applied to the polynomials, where each application eliminates a variable of choice by projection, until all but one variables are eliminated. In the second *lifting phase*, a set of n-dimensional sample points is substituted into the $n+1$-dimensional polynomials from the projection phase. This yields univariate polynomials, whose roots are used to generate new sample points. This phase is repeated in reverse projection order to obtain a set of full-dimensional sample points, one for each region that is sign-invariant w.r.t the input polynomials.

There are numerous optimizations that can be made in the CAD procedure, especially when it is utilized for satisfiability checking of constraint sets instead of full quantifier elimination. In this case, a full CAD is not necessarily needed; instead, sample points can be generated incrementally until a satisfying sample is found and the procedure can be stopped. In this thesis, such an *incremental CAD*, which is optimized for SMT solving, will be considered. In addition to these optimizations, there are heuristic choices that also apply to general CAD. One of these choices is the order in which variables are eliminated, which also dictates the order for the lifting phase. Depending on the structure of the polynomial set, an optimal order can result in projection sets that are smaller in size and degree than those a naive ordering would produce, resulting in a faster CAD computation. This reduction in runtime can be significant in practice, which motivates ongoing research in variable ordering heuristics for CAD.

In this thesis, we will evaluate a recently proposed heuristic based on chordal graphs [LXZZ21] and compare its peformance to a simpler heuristic based on variable degree in the context of an incremental CAD procedure optimized for SMT solving. Furthermore, we will explore how the chordality-based heuristic can be modified to incorporate additional information on the polynomials, which is not currently accounted for in the method. The resulting heuristics will also be tested and compared against the existing methods.

Chapter 2

Preliminaries

2.1 Definitions

We will use the symbols $\mathbb{R}$ for the real numbers, $\mathbb{C}$ for the complex numbers and $\mathbb{N}$ for the natural numbers, assuming $0 \in \mathbb{N}$. For a set A, we define $\mathcal{P}(A) := \{A' \mid A' \subseteq A\}$ as the power set of A.

We will use tuples to represent ordered sequences of elements, as in $\alpha = (a_1, a_2, ..., a_n)$. Assuming $\alpha = (a_1, ..., a_n), \beta = (b_1,...,b_n)$, we will define $\alpha \circ \beta := (a_1,...,a_n, b_1,...,b_n)$ as the concatenation of two tuples. With A being a set and α being a sequence, we define $A \setminus \alpha = \{a \in A \mid a \text{ does not appear in } \alpha\}$. We define $\alpha^n := \underbrace{\alpha \circ \cdots \circ \alpha}_{n \text{ times}}$ as the n-fold concatenation of α with itself.

We will frequently use tuples in algorithms, where we apply the following syntax: $\alpha \leftarrow (a_1, ..., a_n)$ assigns the tuple of the values $a_1, ..., a_n$ to the variable α. The expression $(a_1,...,a_n) \leftarrow \beta$ assumes that β is an expression that evaluates to a tuple $(b_1,...,b_n)$ with n elements, and is equivalent to the sequence of expressions $a_1 \leftarrow b_1; a_2 \leftarrow b_2; ...; a_n \leftarrow b_n$.

In the following section, we will introduce the CAD procedure, which works on multivariate polynomials. Therefore, we will now introduce some necessary definitions:

Definition 2.1.1 (Basic polynomial properties). *Let $p := c_n \cdot x^n + c_{n-1} \cdot x^{n-1} + ... + c_1 \cdot x + c_0 \in R[x]$ with $c_n \neq 0$ be a polynomial over a ring R. We define:*

- *The* coefficient set *of p is defined as* $\mathrm{coeff}(p) := \{c_0, ..., c_n\} \setminus \{0\}$
- *The* leading coefficient *of p is defined as* $\mathrm{lc}(p) := c_n$
- *The* degree *of p is defined as* $\deg(p) := n$

Definition 2.1.2 (Univariate interpretation). *For a multivariate polynomial $p \in \mathbb{R}[x_1,...,x_n]$ over a ring R we define $p[x_i] \subseteq R[x_1,...,x_{i-1}, x_{i+1}, x_n][x_i]$ as the interpretation of p as a univariate polynomial in x_i for $1 \leq i \leq n$ with coefficients from the polynomial ring $R[x_1,...,x_{i-1}, x_{i+1}, x_n]$.*

Definition 2.1.3 (Multivariate polynomial properties). *For a multivariate polynomial $p \in \mathbb{R}[x_1,...,x_n]$ over a ring R we define:*

- *The* degree *of p w.r.t x_i as* $\deg(p, x_i) = \deg(p[x_i])$
- *The* variable set *of a polynomial* $\mathrm{var}(p) := \{x_i \mid deg(p, x_i) > 0, 1 \leq i \leq n\}$

Definition 2.1.4 (Total degree)**.** *The* total degree *of a monomial $m = \prod_{i=1}^{n} x_i^{e_i} \in R[x_1, ..., x_n]$ over a unique factorization domain R with $e_i \in \mathbb{N} \setminus \{0\}$ is defined as* $\mathrm{tdeg}(m) = \sum_{i=1}^{n} e_i$

2.2 The CAD Procedure

The Cylindrical Algebraic Decomposition method was first proposed by Collins to solve the problem of *quantifier elimination* for non-linear real arithmetic [Col75]. For an input polynomial set $P \subseteq \mathbb{R}[x_1,...,x_n]$, the CAD procedure generates a decomposition of $\mathbb{R}^n$ into finitely many *cells*, which are P-sign-invariant regions over $\mathbb{R}^n$. To obtain this decomposition, it goes through two phases:

In the Projection phase, a *projection operator* is applied iteratively to generate polynomial sets in fewer variables, until we arrive at univariate polynomials. Formally, a projection operator is a map of the type $\mathrm{Proj} : \mathcal{P}(\mathbb{R}[x_1,...,x_{k+1}]) \rightarrow \mathcal{P}(\mathbb{R}[x_1,...,x_k])$. So, by applying the operator to our set of input polynomials $P_n := P \subseteq \mathbb{R}[x_1,...,x_n]$, we receive a set $P_{n-1} \subseteq \mathbb{R}[x_1,...,x_{n-1}]$. Ultimately, we will arrive at a set of univariate polynomials $P_1 \subseteq \mathbb{R}[x_1]$. With these univariate polynomials, we transition to the *lifting phase.* We take their real roots, and from them generate a set S_1 of *sample points*, which contains of the following:

- the roots of the univariate polynomials themselves
- one point between each pair of adjacent roots and
- a point below the lowest root and above the highest root, respectively.

Since P_1 consists of univariate polynomials, it is easy to see that the sign-invariant regions consist exactly of the roots, the open intervals between two adjacent roots, and the open intervals between the outermost roots and ∞ resp. $-\infty$. Thus, we have successfully chosen a sample point for each sign-invariant region of P_1.

Now, the algorithm 1 can be performed iteratively, until we get a full-dimensional set of sample points S_n.

As noted, it is easy to see that S_1 contains sample points from all sign-invariant regions of P_1. However, the higher-dimensional generalization "S_k contains sample points of all sign-invariant regions of P_k" is a lot harder to prove. Here, the completeness of the CAD depends on properties of the used projection operator. There are various established projection operators for which the completeness of the CAD procedure has been proven [Col75], [McC98], [Bro01]. Since we are not interested in experiments with the projection operator, we will not go into more details about their completeness.

In general, the runtime of CAD algorithms, even with modern optimizations, is doubly exponential: For the classic sign-invariant CAD, Brown and Davenport ([BD07], Theorem 8) have proven that a complete CAD can contain at least 2^{2^n} cells in the worst case, where n is the number of variables. They have also shown that the variable ordering has a large influence on the runtime: For extreme cases, the cell count for the same input problem can range from linear to doubly exponential ([BD07], Theorem 7). This shows that it can be very worthwhile to try to optimize

Algorithm 1 Lift samples by one dimension

```
procedure LIFTING($S_n$)
    $S_{n+1} \leftarrow \emptyset$
    $R \leftarrow \emptyset$
    for all sample points $(a_1, ..., a_n) \in S_n$ do
        for all polynomials $p \in P_{n+1}$ do
            $\tilde{p} \leftarrow p(a_1,...,a_n)$                          ▷ $\tilde{p}$ is univariate in $x_{n+1}$
            $R \leftarrow R \cup \{a_{n+1} \mid \tilde{p}(a_{n+1}) = 0\}$            ▷ Calculate real roots
        end for
        $S_{n+1} \leftarrow S_{n+1} \cup \{(a_1,...,a_n, a_{n+1}) \mid a_{n+1} \in R\}$     ▷ Sample the roots
        for all open intervals $I$ between two adjacent roots from $R$ do
            choose $a_{n+1} \in I$
            add $(a_1,...,a_n, a_{n+1})$ into $S_{n+1}$
        end for
        choose $a^{<}_{n+1}$ smaller than all roots and $a^{>}_{n+1}$ bigger than all roots in $R$
        add $(a_1,...,a_n, a^{<}_{n+1})$ and $(a_1,...,a_n, a^{>}_{n+1})$ into $S_{n+1}$
    end for
    return $S_{n+1}$
end procedure
```

the variable ordering, even if we have to solve hard problems in our optimization process.

2.2.1 Projection Operators

In this section, we will introduce McCallum's projection operator. It utilizes a combination of polynomial operations, which we will define in the following:

Definition 2.2.1 (Primitive part and content)**.** *Let $p \in R[x]$ be a polynomial over the unique factorization domain R.*

- *The* content *of p is defined as* $\operatorname{cont}(p) = \gcd(\operatorname{coeff}(p))$
- *The* primitive part *of p is defined as* $\operatorname{prim} p = \frac{p}{\operatorname{cont}(p)}$

We note that $\mathbb{R}$ is a unique factorization domain and for any unique factorization domain R, the polynomial ring $R[x]$ is a unique factorization domain as well. Thus, the above definitions are valid for multivariate polynomials over the reals interpreted as univariate polynomials with polynomial coefficients.

Definition 2.2.2 (Operations on polynomial sets)**.** *For a set $P \subseteq R[x]$ of univariate polynomials over a unique factorization domain R we define* $\operatorname{op}(P) = \{\operatorname{op}(p) \mid p \in P\}$ *for* $\operatorname{op} \in \{\operatorname{lc}, \operatorname{prim}, \operatorname{cont}\}$*, and* $\operatorname{op}(P) = \bigcup_{p \in P} \operatorname{op}(P)$ *for* $\operatorname{op} \in \{\operatorname{coeff}\}$

For a set $P \subseteq R[x_1,...,x_n]$ of multivariate polynomials over a ring R, we define $P[x_i] = \{p[x_i] \mid p \in P\}$ for x_i with $1 \leq i \leq n$. We define $\operatorname{var}(P) = \bigcup_{p \in P} \operatorname{var}(p)$.

Definition 2.2.3 (Sylvester matrix, resultant and discriminant)**.** *Let $p := \sum_{i=0}^{n} a_i \cdot x^i \in R[x], q := \sum_{i=0}^{m} b_i \cdot x^i \in R[x]$ be polynomials with $m := deg(p), n := deg(q)$ and $m + n \geq 1$ over a ring R. The associated* Sylvester Matrix $\operatorname{Syl}_{p,q}$ *is a matrix over*

$R^{(m+n)\times(m+n)}$. *The rows of the matrix are defined as* $(\alpha_0^p,,...,\alpha_{n-1}^p, \alpha_0^q, ..., \alpha_{m-1}^q)$, *where:*

$$\alpha_i^p = (0)^i \circ (a_0, ..., a_m) + (0)^{m-i+1}$$
$$\alpha_i^q = (0)^i \circ (b_0, ..., b_n) + (0)^{n-i+1}$$

Now, we can define

- *The* Resultant $\text{res}(p,q) := |\, \text{Syl}_{p,q} \,| \in R$
- *The* Discriminant $\text{disc}(p) := (-1)^{\frac{n(n-1)}{2}} \frac{\text{res}(p,p')}{\text{lc}(p)} \in R$

Example 2.2.1. *Let* $p = \sum_{i=0}^4 a_i x^i, q = \sum_{i=0}^3 b_i x^i$. *Then,*

$$\text{Syl}_{p,q} = \begin{bmatrix} a_4 & a_3 & a_2 & a_1 & a_0 & 0 & 0 \\ 0 & a_4 & a_3 & a_2 & a_1 & a_0 & 0 \\ 0 & 0 & a_4 & a_3 & a_2 & a_1 & a_0 \\ b_3 & b_2 & b_1 & b_0 & 0 & 0 & 0 \\ 0 & b_3 & b_2 & b_1 & b_0 & 0 & 0 \\ 0 & 0 & b_3 & b_2 & b_1 & b_0 & 0 \\ 0 & 0 & 0 & b_3 & b_2 & b_1 & b_0 \end{bmatrix}$$

Definition 2.2.4 (Finest square-free basis [Col75]). *For a set A of polynomials, the* finest square-free basis *of A,* $\mathcal{F}(A)$ *is defined as the set of all the irreducible factors of the elements of A*

Now, we are able to define the McCallum projection operator, which is used for all projection operations unless noted otherwise:

Definition 2.2.5 (McCallum Projection operator [LXZZ21] [McC98]). *For* $A \subseteq \mathbb{R}[x_1,...,x_n]$, *some* $1 \leq i \leq n$ *and* $B := \mathcal{F}(\text{prim}(A[x_i]))$, *we define*

$$\text{Proj}_{mc}(A, x_i) = \text{cont}(A[x_i]) \cup \bigcup_{f,g \in B, f \neq g} \{\text{res}(f[x_i], g[x_i])\} \cup \bigcup_{f \in B} (\text{coeff}(f[x_i]) \cup \{\text{disc}(f[x_i])\})$$

When the variable for the projection is clear from the context, we will sometimes omit the variable and just write $\text{Proj}_{mc}(A)$

This operator offers some improvements over the original projection operator given by Collins when he first presented the CAD procedure in [Col75]. McCallum's operator does allow for a better bound on the theoretical worst-case runtime [McC98] and, in general, tends to perform better in practice in an SMT-solving context [VKÁ07]. However, the operator is incomplete: If used as a direct replacement for Collins' operator, the resulting CAD might not contain a sample point for all sign-invariant regions. If used in an NRA solver, it might falsely deduce UNSAT for a constraint set that is indeed satisfiable, because the satisfying sample point was not generated using the incomplete projection. To obtain a complete CAD using McCallum's operator, the input polynomials must be *well-oriented* ([McC98], definition on page 92). Most polynomial sets are well-oriented, in particular all sets of polynomials with no more than three variables. In practice, checks can be introduced into the lifting procedure to detect whether the input polynomial set was, at some point, not well-oriented. In

Table 2.1: The (m,d)-property of the projection sets [LXZZ21],[BDE+16]

# variables	Number	Degree
n	m	d
$n-1$	M	$2d^2$
$n-2$	M^2	$8d^4$
$n-3$	M^3	$128d^8$
$\vdots$	$\vdots$	$\vdots$
$n-r$	$M^{2^{r-1}}$	$2^{2^r-1}d^{2^r}$
$\vdots$	$\vdots$	$\vdots$
1	$M^{2^{n-1}}m$	$2^{2^n-1-n}d^{2^n-1}$

this case, the procedure can be aborted and the caller of the CAD procedure can decide how to deal with the situation (e.g. by invoking CAD with Collins' operator). However, having a non-well-oriented polynomial set is sufficiently rare, so in most cases the additional effort of checking and possibly re-computing the CAD is outweighed by the performance gains of this operator (and derived ones) over complete operators like Collins'.

2.2.2 Complexity Analysis

In this section, we will give a short overview of the complexity analysis for CAD that was originally given in [BDE+16]. In [LXZZ21], an improved lower bound is obtained with a slight adaptation of the method for CAD using the proposed ordering heuristic.

Definition 2.2.6 (Combined degree and (m,d)-property [BDE+16])**.** *The* combined degree *of a polynomial set $A \in \mathbb{R}[x_1,...,x_n]$ is defined as* $\max_{x \in \text{var}(A)} \deg(\prod_{a \in A} a, x)$*, the maximal degree w.r.t. any variable of the product of the polynomials. The set A has the* (m,d)-property *if it can be partitioned into at most m pairwise disjoint subsets, each with combined degree at most d*

In their complexity analysis, Bradford et al. used the (m,d)-property as a measure of the size of the projection polynomial sets. To perform this analysis, they proved the following lemma for the McCallum projection operator: Let $A \subseteq \mathbb{R}[x_1,...,x_n]$ be a polynomial set with the (m,d)-property. Then, the set $\text{Proj}_{mc}(A)$ will have the (M,$2d^2$) property with $M = \lfloor \frac{(m+1)^2}{2} \rfloor$. With this relation, it is already visible that the repeated application of the projection operator means that a "power tower" is constructed in the terms for expressing the (m,d)-property, which can be written out as a double exponential term in the general case. Table 2.1 shows the growth of the terms in the (m,d)-property.

Now, these (m,d)-properties can be used to obtain an upper bound on the number of CAD cells generated for an univariate polynomial set: A polynomial set with the (m,d)-property has at most m polynomials, which, in turn, can have at most d zeroes. Since the sample points for a single polynomial are constructed from the zeroes themselves, points between the zeroes, and two points lower resp. higher than all zeroes, we obtain an upper bound of $2md + 1$. With this, the following upper bound on the number of CAD cells can be deduced [LXZZ21]:

Theorem 2.2.1 ([LXZZ21]). *The number of CAD cells in $\mathbb{R}^n$ obtained by the Projection operator* Proj_{mc} *is* $O\left((2d)^{2^n-1}M^{2^{n-1}-1}m\right)$ *with* $M = \lfloor\frac{(m+1)^2}{2}\rfloor$.

We will accept this upper bound on the number of cells as a rough estimate for the runtime of the CAD algorithm.

2.3 Chordal Graphs

The main work this thesis is based on uses *graphs* to track connections between variables in polynomials. Based on these connections, a variable ordering is devised. Formally, we encode these connections in an *undirected graph*

Definition 2.3.1 (Undirected Graph). *An* undirected graph *is a pair $G = (V, E)$ of a finite set V of* vertices *and a set $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$ of* edges.

Definition 2.3.2 (Path, Connected Graph). *A sequence of at least two vertices $(v_1, v_2, ..., v_n)$ is a path in an undirected graph (V, E) if there are edges $\{v_i, v_{i+1}\} \in E$ for $1 \leq i < n$.*

Definition 2.3.3 (Connected components). *For an undirected graph $G = (V, E)$ we define the set of* connected components *of the graph as a partition of $V = V_1 \dot\cup ... \dot\cup V_n$ into disjunct subsets of V, where two vertices $v_1, v_2 \in V_i$ are in the same connected component if there is a path $(v_1, ..., v_2)$ in G. A graph with a single connected component is* connected

Definition 2.3.4 (Adjacency and incidence). *Let $G = (V,E)$ be an undirected graph. We say that v is* adjacent *to v', if $\{v, v'\} \in E$. Accordingly, we can define the* adjacency *or* neighborhood *of v as $adj(v) := \{v' \mid \{v, v'\} \in E\}$. An edge $e \in E$ is* incident to *v, if $v \in e$. Accordingly, we define* $\text{inc}(v) = \{e \in E \mid v \in e\}$ *as the set of edges that is incident to v.*

Now, we define some interesting sub-structures of a graph that are helpful for our analysis:

Definition 2.3.5 (Clique, Simplical vertex). *A* clique *$C \subseteq V$ is a set of vertices that is pairwise connected, so for every $\{v, v'\} \subseteq V$ with $v \neq v'$, there is also $\{v, v'\} \in E$. A vertex $v \in V$ is* simplical, *if the set* $\text{adj}(v)$ *of adjacent vertices forms a clique.*

Definition 2.3.6 (Cycle). *A* cycle *is a sequence $(v_1,...,v_n)$ of vertices $v_i \in V, 1 \leq i \leq n$ so for every $1 \leq i < n$, $\{v_i, v_{i+1}\} \in E$, and $\{v_n, v_1\} \in E$.*

Definition 2.3.7 (Chordal Graph). *A graph is* chordal, *if every cycle $(x_1, x_2, ..., x_n)$ of length greater than three has a* chord, *that is, there exists an edge $\{x_i, x_j\} \in E$ such that $\{i,j\} \notin \{\{k, k+1\} \mid k \in \{1, ..., n = 1\}\} \cup \{\{1, n\}\}$.*

2.3.1 Graphs for Variable Elimination

Research has been done on the use of graphs to model the elimination of variables in mathematical procedures. In 1961, Parter proposed a heuristic for ordering the variable in the Gaussian elimination procedure for linear systems of equations [Par61]. Much later in 2021, Li et al. [LXZZ21] present a variable ordering for CAD which is roughly based on the same idea. Since this variable ordering will be our primary focus, we present some research that has been done in the field of variable elimination procedures using chordal graphs.

Parter was concerned with the optimization of Gaussian elimination for linear systems of equations of the form $Ax = y$, where A is an $n \times n$ matrix and x and y are column vectors. It is well known that the elimination can be done easier if A is sparse, however, sparseness is not a very rigorously defined property. The contribution of this work was a deeper understanding of sparseness, which was reached by generating an associated graph $G(A)$ for a given matrix A:

The graph contains n vertices $V := \{1, 2, ..., n\}$ for an $n \times n$ matrix, where $\{i,j\} \in E \iff a_{i,j} \neq 0$.

With this graph structure, it is possible to simulate the effect of Gaussian elimination on the matrix structure by eliminating vertices in the graph instead of variables in the matrix. This process can be formalized using the *elimination game* ([Par61], Theorem 1):

```
procedure ELIMINATE($G = (V, E), v \in V$)
    $F \leftarrow \emptyset$                                    ▷ The set of fill edges
    for all pairs $\{v_1, v_2\} \subseteq \mathrm{adj}(v)$ do
        if $\{v, v'\} \notin E$ then
            $F \leftarrow F \cup \{\{v_1, v_2\}\}$
            $E \leftarrow E \cup \{\{v_1, v_2\}\}$
        end if
    end for
    $V \leftarrow V \setminus \{v\}$                             ▷ Remove the vertex
    $E \leftarrow E \setminus \mathrm{inc}(v)$                        ▷ Clean up the edges
    return $G = (V, E), F$
end procedure
```

This algorithm simulates the successive elimination of variables in the following sense: When a variable x_i is eliminated in the linear system A, the associated graph $G(A')$ of the new matrix A' is a subgraph of the graph returned by ELIMINATE$(G(A), x_i)$.

In a sense, this theorem gives an "upper bound" on the set of non-zero matrix entries after an elimination step by using graph theory. Now, this allows us to study the sparseness of a matrix not only in its initial form, but through the whole elimination process. For this, the elimination algorithm has to be applied successively, resulting in the following algorithm:

Algorithm 2 Elimination-Game [Par61]

```
procedure ELIMINATION-GAME($G = (V, E), \alpha$)        ▷ $\alpha$ is a sequence of vertices
    $F \leftarrow \emptyset$                                    ▷ The set of fill edges
    while $V \neq \emptyset$ do
        $v \leftarrow$ POPFRONT($\alpha$)       ▷ Removes the first element of $\alpha$ and returns it
        $(G, F') \leftarrow$ ELIMINATE(G, v)
        $F \leftarrow F \cup F'$
    end while
    return $F$
end procedure
```

For the runtime of the elimination algorithm, it is ideal to keep the matrix as sparse as possible (so fewer operations have to be calculated). Hence, the elimination procedure should eliminate the variables in an order that introduces no fill-in in the elimination game algorithm (and therefore, no additional fill-in in the matrix). Such an ordering is also called a *perfect elimination ordering*. More rigorously, this can be defined as follows:

Definition 2.3.8 (Perfect elimination ordering [LXZZ21]). *Let $G = (V,E)$ be an undirected graph. An order of vertices, $v_1 < v_2 < ... < v_n$ with $\{v_1, ..., v_n\} = V$ is a* perfect elimination order (peo)*, if for every vertex $v \in V$, the set*

$$X_v := \{v\} \cup \{v' \mid v' > v, \{v, v'\} \in E\}$$

forms a clique in G.

We will commonly define elimination orders using ordered sequences $(v_1, ..., v_n)$, in which case the order $i < j \Leftrightarrow v_i < v_j$ is implied. Also, note that the definition in [LXZZ21] writes $v > v'$ if v is eliminated before v', while we write this case as $v < v'$. This allows us to obtain a list with the vertices in the elimination order by sorting the set V ascending by $<$, which is the convention used by most sorting algorithms in the C++ standard library. It has been proven by Fulkerson [FG65] that the class of graphs that have perfect elimination orderings is exactly the class of chordal graphs.

To optimize Gaussian elimination using this theory, we require an algorithm that can find a perfect elimination ordering for a given graph. A very simple procedure for this task is based on the successive removal of *simplical vertices*. A vertex v is *simplical* if $\text{adj}(v)$ forms a clique. Therefore, eliminating a vertex introduces no fill in the elimination game iff the vertex is simplical. Now, we can define our procedure as follows:

```
$\alpha_{peo} = ()$
while there is a simplical vertex $v$ in the Graph do
    $\alpha_{peo} \leftarrow \alpha peo \circ (v)$
    Remove $v$ from $V$ and $\text{adj}(v)$ from $E$
end while
if $V = \emptyset$ then
    return $\alpha_{peo}$    ▷ The whole graph was eliminated with no fill, so we found a peo
else
    abort    ▷ There are vertices still in the graph, but none is simplical, so any
elimination will introduce fill. Thus, the graph is non-chordal
end if
```

The correctness of the algorithm follows from two basic statements about chordal graphs: First, any chordal graph has at least one simplical vertex (otherwise, there would be no peo - choosing any vertex as the first would introduce fill). Furthermore, any induced subgraph of a chordal graph is also chordal. To give a short proof for this statement, we will assume $G = (V, E)$ is a chordal graph, and we remove one vertex v together with its incident edges. Now, remember that a graph is chordal iff any cycle of length at least four has a chord. Thus, the only way to make the graph non-chordal is by either adding a new cycle, or removing a chord. Since we do not add any edges, no new cycle will be added. However, consider now that v is incident to a chord of a

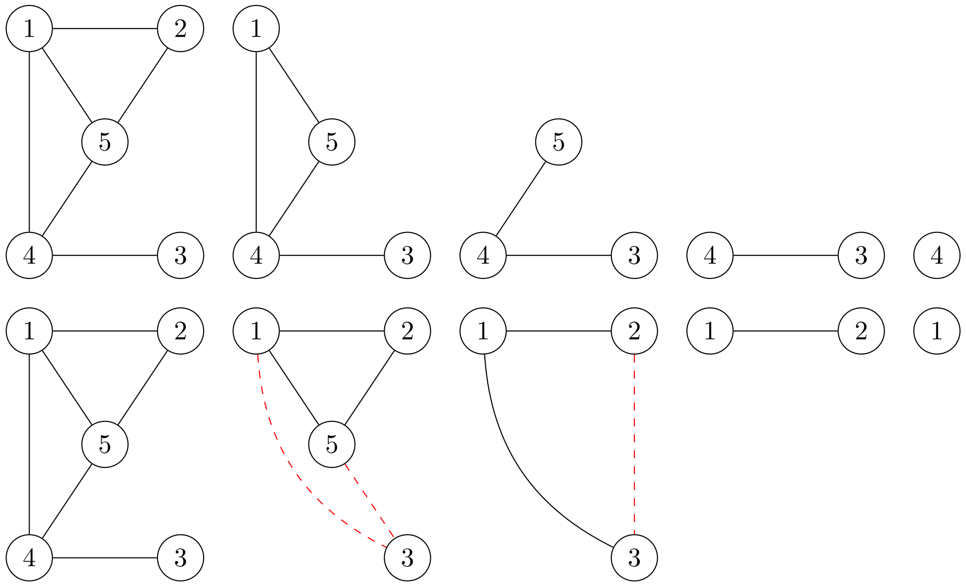


Figure 2.1: The elimination game played on a chordal graph. On the top row, we use the peo $2 < 1 < 5 < 4 < 3$. On the bottom row, the non-perfect ordering $4 < 5 < 3 < 2 < 1$ is used, which introduces fill edges. In each step, the new fill edges are dashed and colored red.

cycle. Per definition of a chord, v is part of the cycle. Thus, by removing v, the cycle is broken. Therefore, the subgrapy induced by $V \setminus \{v\}$ is chordal. Per induction, the statement follows for all induced subgraphs.

While this algorithm is very simple, repeatedly checking for simplical vertices is expensive. There are better algorithms for finding peos, like the Maximum Cardinality Search (MCS) algorithm, first presented by Tarjan and Yannakakis [TY84]. The algorithm runs in linear time $O(|V| + |E|)$.

Even if the graph is non-chordal, it could be 'nearly chordal' in the sense that the minimum set of fill edges required to make the graph chordal is very small. In this case, adding the fill edges and then choosing a perfect elimination ordering can still have an advantage compared to a completely naive ordering. Hence, we will introduce some tools to deal with *chordal completions* of graphs:

Definition 2.3.9 (Chordal completion)**.** *Let $G = (V,E)$ be a non-chordal graph. $F \subseteq \{\{v, v'\} \mid v \in V\} \setminus E$ is a set of* fill edges *that are not already present in the graph. We call $G' = (V, E \cup F)$* a chordal completion *of G iff G' is chordal.*

We call $G' = (V, E \cup F)$ a minimal chordal completion *of G iff it is a chordal completion of G and the graph $G^- = (V, E \cup F^-)$ is not chordal for any $F^- \subsetneq F$, i.e. if we take away any subset of the fill edges, the graph is no longer chordal.*

We call $G' = (V, E \cup F)$ a minimum chordal completion *of G iff it is a chordal completion of G and for all other chordal completions $G'' = (V, E'')$ we have that $|E'| \leq |E''|$.*

The MCS-M algorithm (see algorithm 3) was first presented by Berry et al. [BBHP04] to find minimal chordal completions. It is a simplification of another well-known algorithm for the same purpose (LEX-M). For an input graph $G = (V, E)$, the algorithm runs in $\mathcal{O}(|V| \cdot |E|)$, so $\mathcal{O}(n^3)$, where n is the number of vertices.

Algorithm 3 MCS-M [BBHP04]

```
procedure MCS-M($G = (V,E)$)                                    ▷ $G$ is non-chordal graph
    $F \leftarrow \emptyset$                                    ▷ The set of fill edges
    $\alpha \leftarrow ()$                                      ▷ The elimination ordering
    for all vertices $v \in V$ do
        $w(v) \leftarrow 0$                                     ▷ Initialize the weight map $w : V \to \mathbb{N}$
    end for
    for $i \leftarrow n$ downto 1 do
        Choose an vertex $v \in V \setminus \alpha$ of maximum weight $w(v)$
        $S \leftarrow \emptyset$
        for all unnumbered vertices $u \in G$ do
            if $\{u, v\} \in E$ or there is a path $(u, x_1, x_2, ...x_k, v)$ through $\{x_1,...,x_k\} \subseteq V \setminus \alpha$ such that $w(x_i) < w(u)$ for $1 \leq i \leq k$ then
                $S \leftarrow S \cup \{u\}$
            end if
        end for
        for all vertices $u \in S$ do
            $w(u) = w(u) + 1$
            if $\{u,v\} \notin E$ then
                $F \leftarrow F \cup \{\{u,v\}\}$
            end if
        end for
        $\alpha \leftarrow (v) \circ \alpha$                    ▷ Prepend $v$ to the elimination ordering
    end for
    return $(\alpha, F)$      ▷ Returns the computed fill edges and the elimination order. The chordal completion can be computed as $G^+ = (V, E \cup F)$
end procedure
```

The problem of finding a *minimum* chordal completion is NP-complete, so assuming $\mathsf{P} \neq \mathsf{NP}$, the worst-case runtime of an algorithm for this task is exponential.

2.3.2 Elimination Trees

The elimination tree is an associated tree structure that can be defined for a peo, which holds information about the relation of the vertices to be eliminated.

Definition 2.3.10 (Elimination Tree ([LXZZ21], Definition 5.3))**.** *Let $G = (V,E)$ be a graph, with $x_n < x_{n-1} < ... < x_1$ being a perfect elimination order on that graph. The* elimination tree *of G is a directed spanning tree $T = (V, B)$ with the following edge relation: For every x_i with $i \neq n$, B contains a single edge (x_i, x_p), where $x_p = \min\{x \in \operatorname{adj}(x_i) \mid x > x_i\}$, which we will call the* parent *of x_i. We define the children $\operatorname{child}(x_p) = \{x_l \mid (x_l, x_p) \in B\}$ of a vertex accordingly. The height $h(x)$ of a vertex x is defined as 1 if the vertex is a leaf, and $\max\{h(x') \mid x' \in \operatorname{child}(x)\} + 1$ Note that x_1 has no outgoing edges, and is therefore the root of the elimination tree.*

When working with the elimination tree, we will often consider the following lemma:

Lemma 2.3.1 ([LXZZ21], Lemma 5.4)**.** *Let $T = (V, B)$ be the elimination tree of a chordal graph G. If $x_s < x_t$ and $(x_s, x_t) \in B$, then there is a path from x_t to x_s in T*

Its contraposition is particularly interesting: If there is no path between two nodes (i.e. they are on two different branches of the tree), then there is also no edge between them in the original graph. This property of elimination trees was also recognized by Jess and Kees [JK82], who used elimination trees for parallel L/U Decomposition of linear systems (which is very similar to gaussian elimination). Hence, the same ideas from [Par61] can be applied for single-threaded variants of the algorithm. To execute multiple eliminations in parallel however, it is important to ensure that there are no conflicts on shared memory: If one thread of the algorithm reads out a coefficient in the matrix while it is being overwritten by another thread, the result will be inconsistent. In the case of Jess and Kees, such a conflict would occur if and only if two variables were eliminated that form an edge on the associated graph. Thus, parallel elimination is possible if the algorithm takes care to never eliminate any two variables at the same time that have an edge.

With these constraints, Jess and Kees wanted to optimize the runtime of the algorithm, assuming no restrictions on the maximum number of parallel threads. To do this, they also introduced the same concept of an elimination tree, which they used in their work as a kind of generalization of perfect elimination orderings for parallel elimination: In a single step, the decomposition algorithm would eliminate all leaves of the elimination tree - since no two leaves can have a path between them, this is always safe. With this strategy, the time until completion is given by the height of the elimination tree. Hence, finding an elimination tree with minimum height for a given graph helps to optimize the runtime of the algorithm. Later, we will introduce a variable ordering heuristic for CAD that makes use of the elimination tree, so we will now describe the algorithm by Jess and Kees, which is given in 4.
Since they were concerned with parallel optimization, they did not use the notion of a perfect elimination ordering as it was introduced before. Instead, they assign a *class label* to each vertex (which is denoted as $\gamma(v)$), which can be used to obtain an order of vertex classes $X_1, X_2, ..., X_k$ where $X_i = \{v \mid \gamma(v) = i\}$. These label classes give the equivalent of a peo for parallel algorithms: Any elimination order $v_1 < v_2 < ... < v_n$, where $v < v' \implies \gamma(v) <= \gamma(v')$, is a perfect elimination ordering. The algorithm E-TREE works by first assigning such labels to the vertices, and then constructing an elimination tree based on these labels. This elimination tree then conforms to for any peo that can be derived from the label classes as described above.

Note that we make two minor modifications to the procedure: First, in the inner loop, we discard all vertices with non-zero deficiency first before performing the elimination procedure for one of the leftover vertices (which now has zero deficiency and therefore allows elimination without fill-in). This modification allows us to record the number of zero-deficiency vertices in each step as a number of 'choices', which can have an influence on the outcome of the final algorithm. As a second addition, the algorithm determines a perfect elimination order that eliminates the vertices ordered by their level in the elimination tree ascending (lower levels eliminated first).

Assuming the algorithm was ran on a connected, chordal graph, the generated elimination tree conforms to the given definition and is of minimum height.

For non-chordal graphs, the result of the E-TREE procedure is undefined. Therefore, Jess and Kess suggest to compute a minimal chordal completion before applying the procedure. However, as shown by Pothen ([Pot88], Theorem 1), the problem of finding the shortest elimination tree on a general (non-chordal) graph is NP-complete.

Algorithm 4 E-Tree [JK82]

```
procedure E-Tree(G = (V, E))
    $(\hat{V}, \hat{E}) \leftarrow (V, E)$                          ▷ Working copy of the graph
    $i \leftarrow 1$                                               ▷ The current level
    $\alpha \leftarrow ()$                                          ▷ A perfect elimination ordering
    while $\hat{V} \neq \emptyset$ do
        $U \leftarrow V$
        while $U \neq \emptyset$ do
            while there is a non-simplical $v \in U$ do
                $U \leftarrow U \setminus \{v\}$
            end while
            if $U = \emptyset$ then
                break
            end if
            Choose $v \in U$ ▷ $v$ is guaranteed to be simplical
            $\gamma(v) \leftarrow i$                                ▷ $\gamma(v)$ is the label of $v$
            $U \leftarrow U \setminus \mathrm{adj}(v, \hat{E})$
            $\hat{V} \leftarrow \hat{V} \setminus \{v\}$
            $\hat{E} \leftarrow \hat{E} \setminus \mathrm{inc}(v, \hat{E})$
            $\alpha \leftarrow \alpha \circ (v)$                    ▷ Append $v$ to the peo
        end while
        $i \leftarrow i + 1$
    end while
    $B \leftarrow \emptyset$                                        ▷ $T = (V, B)$ forms the E-Tree
    for all $v \in V$ do
        $l(v) \leftarrow \{\gamma(w) \mid w \in \mathrm{adj}(v, E) \wedge \gamma(w) > \gamma(v)\}$
        $B \leftarrow B \cup \{(v, w) \mid w \in \mathrm{adj}(v, E) \wedge \gamma(w) \in l(v) \wedge \gamma(w)$ is minimum$\})$
    end for
    return $(T = (V, E), \alpha)$
end procedure
```

Thus, assuming $\mathsf{P} \neq \mathsf{NP}$, the best possible general algorithm for finding the shortest e-tree is of exponential complexity, which makes it infeasible to use in the optimization of many problems, since the gains will be outweighed by the long runtime of the E-Tree algorithm.

2.4 SMT Solving

In this thesis, we are concerned with the satisfiability problem for arbitrary logical formulas that are constructed using NRA constraints as atoms. While it is possible to solve such problems using only the CAD procedure, it would require us to compute the CAD on all constraints of the formula and test the truth value of the full formula for each sample point. This approach is not ideal, since it ignores the logical structure of the formula: By analyzing the logical structure of the input problem, we often discover that knowledge of a subset of the constraints is sufficient to prove satisfiability. Therefore, we consider the combination of CAD with a Boolean SAT solver to construct a less-lazy SMT solver. The following section describes the general structure of such a solver to the extent that is useful for later analysis. For a more thorough explanation of SMT solving, the reader is referred to [Kre20].

In its core, the structure of an SMT solver can be described as follows: As a first step, the Boolean abstraction of the input formula is built. In this Boolean abstraction, we replace each arithmetic constraint (of the form $p \sim 0$) with a unique Boolean variable $b_{p\sim 0}$ representing this constraint. Then, the Boolean abstraction is submitted to the SAT solver, which will try to find a satisfying assignment for the abstraction, ignoring the arithmetic parts completely. When the solver found a satisfying assignment for the Boolean abstraction variables, it is necessary to check if it is possible to find an assignment for the *arithmetic* variables, such that the truth values of the constraints match the truth values of the corresponding abstraction variables in the assignment found by the SAT solver. For this step, the CAD procedure can be used. If it returns a satisfying arithmetic assignment, it is also satisfying for the input formula, so the SMT solver can terminate with an answer of SAT. Otherwise, the SAT solver has to mark the assignment as UNSAT and further explore the search space to find a different assignment that might be consistent with the theory. If no such assignment can be found after exploring the full Boolean search space, UNSAT is returned.

2.4.1 CDCL-Style SAT Solving

To fully understand the inner workings of a modern SMT solver, it is helpful to look into the details of CDCL-style SAT solving. While it is possible to consider the SAT solver as a "black box", solving performance on real-world problems can benefit from a tighter integration between Boolean SAT and theory solving.

To keep track of the assignments that have been checked already, the SAT solver uses a *trail*. This is essentially a stack of assignments to single variables with some metadata. A typical solver might solve 2-tuples of the form (L, C), where L is a literal of the form b or $\neg b$ to indicate that b is assigned true resp. false. C can optionally store a clause in the formula. This is used to indicate that, with the current trail, the assignment L is a logical implication of C. Otherwise, C is empty and we call $(L, \square)$ a decision. Now, the solver procedure can be described as the following loop:

1. **Propagation:** Check if there is any assignment that is a logical implication of the current assignments in the trail and the formula. If such an implication exists, store the assignment L together with the implying clause C to the trail. For example, with $\varphi = \neg a \wedge (b \vee c)$ the literal $\neg a$ must be true, so $(\neg a, (\neg a))$ is stored.

   (a) **Backtracking:** If there is a clause C in a formula that is falsified by the current trail, a *conflict* occurred. In this case, the solver has to *backtrack* by removing assignments from the trail in order to revert the decision that led to the conflict. In addition, a process called *Conflict Resolution* is employed to generate a *Conflict Clause* from C and the trail at the point of conflict. This clause is then added to the formula, which prevents the solver from taking the same wrong decision again. After this, we continue with **1: Propagation**

   (b) **Decision:** If there are no more clauses that immediately imply the assignment of another literal, the solver has to "guess" an assignment L, which is called a *decision*. This decision is logged onto the trail as $(L, \square)$, and the solver goes back to **1: Propagation**. If there are no more variables for which a value can be decided, the Boolean formula is satisfied and the solver returns SAT.

If we want to integrate such a solver with a theory solver for SMT solving, we can make some modifications which, in practice, can improve the performance of the solver. First, we can call the theory solver every time before making a decision, instead of waiting until we find a full satisfying assignment. If the theory solver discovers that the current assignment is already inconsistent with the theory, we have saved ourselves from exploring a branch of the search space that contains no solutions for our top-level problems anyways. While this behavior might lead to more theory calls overall in the worst-case, preventing a decision in this way can avoid multiple, even more expensive theory calls that would have been a consequence otherwise. If the theory solver has the ability to return an *infeasible subset* of theory constraints, this can be translated back to its Boolean representation and learned, similar to a conflict clause.

However, there are also modifications on the theory side that can be beneficial for working with a SAT solver: To solve a single SMT problem, the theory solver is potentially called many times, as described above. However, these calls will usually be strongly related to each other: When, after a theory call, the SAT solver is able to propagate without conflict, the next theory call will be a superset of the constraints passed previously. Therefore, it is beneficial if the theory solver can keep the previous state and only incorporate the additional constraints *incrementally* instead of recomputing everything for every call. Similarly, when the SAT solver has to backtrack because of a conflict, the theory solver should be able to *backtrack* by removing constraints from its internal state as well.

In the following sections, we will introduce the Satisfiability Modulo Theories - Real Algebra Toolkit (SMT-RAT) framework, which contains a CDCL SAT solver and a CAD implementation that has the features mentioned above.

2.5 SMT-RAT

SMT-RAT is a "Toolbox for strategic and parallel Satisfiability-Modulo-Theories solving" [smt22] developed by the Theory of Hybrid Systems Group, RWTH Aachen University. Its core part consists of *modules* that offer a standardized interface to solve an SMT-related problem. Two important modules we will be working with are the following:

- The `SATModule`, which solves the Boolean Satisfiability Problem SAT. It implements a CDCL SAT solver as described in Section 2.4.1. This module is utilized almost every time when working with SMT-RAT.
- The `NewCADModule`, which is able to solve QF_NRA constraints incrementally using the CAD method. This work concentrates on implementing and evaluating variable ordering heuristics in the context of this module.

An important concept in SMT-RAT is the composition of modules. A module can take another set of modules as *backends*, which can be called to out-source partial problems. With this composition principle, SMT-RAT can work as a complete SMT solver: The SAT module will receive the input formulae, which consist of theory constraints as atoms, combined with the usual Boolean logic operators. This module will only consider a Boolean logic abstraction of the formula, passing the theory atoms to a backend when they have to be checked for satisfiability. In the setting of QF_NRA, these atoms will be polynomial constraints, which are passed to the `NewCADModule`.

This way, the module structure forms a tree: At the root, the full input problem is inserted, and partial problems are passed down the branches, until the leaves, which are able to solve the problem they are designed for fully on their own. Solutions are then passed back up the tree. This can go back and forth multiple times, until we receive the ultimate answer from our root module. In SMT-RAT, a composition of modules, with the aim of solving some problem, is called a *strategy*. This model allows a user to craft a strategy from the single parts that SMT-RAT offers, which is suited to the particular problem at hand. Listing A.2 on page 57 shows the C++ definition of the strategy outlined above, constructing a simple SMT solver for QF_NRA using a pure CAD backend. In addition to the composition of modules to a strategy, some modules (including the aforementioned `CADModule` and `SATModule`) allow the specification of settings for the module itself, usually by the C++ template mechanism. This high degree of customizability makes SMT-RAT a valuable tool for research in SMT-solving: It is easy to take one of the pre-defined strategies as a baseline and swap out just the part we are interested in for our research (here: the CAD variable ordering, which is specified as a part of the `CADSettings` template parameter).

2.6 CAD Implementation in SMT-RAT

This section contains a brief overview of the `NewCADModule` of SMT-RAT, focusing on the incrementality and backtracking aspects that are typically not found in other CAD implementations. A more thorough explanation can be found in [KA20].

The `NewCADModule` of SMT-RAT is purpose-built for integration into an SMT solver.

As explained in Section 2.4.1, the CAD procedure will be called multiple times, possibly with only slight variations on the constraint set. Considering the doubly exponential complexity of CAD, it would be wasteful to set up a new CAD for every theory call. Instead, the SMT-RAT implementation has the ability to add constraints to build a CAD *incrementally*, or to remove constraints to support *backtracking*.

Another possible optimization in a CAD implementation for SMT solving arises from the observation that we do not need the full CAD. Instead, it is enough to find one satisfying example, i.e. one sample point that satisfies all the constraints that are currently considered. To take advantage of this, the CAD in SMT-RAT is able to perform projection and lifting in a granular fashion and switch between these phases multiple times. This allows the implementation to obtain satisfying samples more quickly in practice.

For example, when called with a set of polynomial constraints, the CAD implementation is able to start with an empty projection and add a single polynomial at a time to the projection set, performing a lifting after every polynomial added. Since we are missing some polynomials, not all sign-invariant regions of the full set are covered, but still enough that we might hit a correct region by chance.

2.7 Simple Variable Ordering Heuristics for CAD

This thesis is focused on a variable ordering based on chordal graphs as proposed by Li et al. [LXZZ21]. However, we will first introduce other variable ordering heuristics that we will use for comparison. First, we will consider some simple heuristics that are based on immediate properties of the polynomials. Hence, these heuristics are very cheap in their application.

2.7.1 Triangular Ordering

This ordering heuristic was used in a CAD-constructing algorithm proposed by Chen et al. [CMXY09]. The algorithm does not follow the usual structure of a projection of lifting that was introduced before. Instead, their algorithm consists of three main steps: First, a decomposition of $\mathbb{C}^n$ is computed. In a second step, this initial decomposition is transformed to be cylindrical (though still over $\mathbb{C}^n$). As a last step, this composition is used to compute a CAD (in the same sense as defined in our introduction) in real space. Since this implementation of a CAD construction is quite different from ours, it is difficult to argue how this heuristic can optimize the projection and lifting process. This is the variable ordering heuristic which is currently implemented in SMT-RAT. However, it is "not clear whether it is a particularly good heuristic for a regular CAD Projection" [Kre20]. We will give a definition of the heuristic as described by England et al. [EBDW14].

Definition 2.7.1 (Triangular ordering)**.** *Let $P \subseteq \mathbb{R}[x_1,...,x_n] \setminus \{0\}$ be a polynomial set over the reals. Define furthermore $t_1, t_2, t_3 : \{x_1,...,x_n\} \to \mathbb{N}$ as follows:*

$$\begin{aligned} t_1(v) &= \max\{\deg(f, v) \mid f \in P\} \\ t_2(v) &= \max\{\text{tdeg}(\text{lc}(f,v)) \mid f \in P \text{ and } v \in \text{var}(f)\} \\ t_3(v) &= \sum_{f \in P} \deg(f, v) \end{aligned}$$

for all $v \in \{x_1,...,x_n\}$ Then, we define the ordering over $\{x_1,...,x_n\}$ as

$$\begin{aligned} x_i > x_j :\iff & t_1(x_i) > t_1(x_j) \\ & \vee t_1(x_i) = t_1(x_j) \wedge t_2(x_i) > t_2(x_j) \\ & \vee t_1(x_i) = t_1(x_j) \wedge t_2(x_i) = t_2(x_j) \wedge t_3(x_i) > t_3(x_j) \end{aligned}$$

for all $i, j \in \{1,...,n\}$.

2.7.2 Brown's Ordering

In his tutorial notes about CAD [BR04], Brown gives a simple variable ordering heuristic, similar to the "triangular" one given above. It is based on the idea of eliminating variables with lesser occurrences first, i.e. those that "appear in few terms and to low degree in the input polynomials". We can define this order as follows:

Definition 2.7.2 (Brown's Ordering)**.** *Let $P \subseteq \mathbb{R}[x_1,...,x_n] \setminus \{0\}$ be a polynomial set* *over the reals. Define furthermore* $b_1, b_2, b_3 : \{x_1,\ldots,x_n\} \to \mathbb{N}$ *as follows:*

$$\begin{aligned}
t_1(v) &= \max\{\deg(f,v) \mid f \in P\}\\
t_2(v) &= \max\{\operatorname{tdeg}(t) \mid t \textit{ is a monomial of a polynomial in } P, v \in \operatorname{var}(t)\}\\
t_3(v) &= |\{t \mid t \textit{ is a monomial of a polynomial in } P, v \in \operatorname{var}(t)\}|
\end{aligned}$$

for all $v \in \{x_1,\ldots,x_n\}$*. Then, we define the ordering over* $\{x_1,\ldots,x_n\}$ *as*

$$\begin{aligned}
x_i > x_j :\Longleftrightarrow\ & b_1(x_i) > b_1(x_j)\\
&\vee b_1(x_i) = b_1(x_j) \wedge b_2(x_i) > b_2(x_j)\\
&\vee b_1(x_i) = b_1(x_j) \wedge b_2(x_i) = b_2(x_j) \wedge b_3(x_i) > b_3(x_j)
\end{aligned}$$

for all $i, j \in \{1,\ldots,n\}$*.*

2.8 Chordality-based Ordering for CAD

In this section we present a variable ordering based on perfect elimination orderings, which was first introduced in [LXZZ21].

The idea presented there is similar to the one in [Par61], where we construct a graph from the input problem, and devise an elimination order of the variables in the input problem from an elimination order in the graph. Essentially, the elimination of variables in the graph with Elimination-Game serves as a very rough approximation of the projection process. Therefore, we will first introduce our graph structure:

Definition 2.8.1 (The associated graph for a polynomial set ([LXZZ21], Def. 2.1))**.** *Let* $P \subseteq \mathbb{R}[x_1,\ldots,x_n]$ *be a set of multivariate polynomials. The* associated graph $\mathcal{G}(P)$ *is defined as* $\mathcal{G}(P) = (V,E)$ *with*

$$V = \operatorname{var}(P), \qquad E = \bigcup_{f \in P} \{\{x_i, x_j\} \mid x_i, x_j \in \operatorname{var}(f), x_i \neq x_j\}$$

Thus, to form the *associated graph* for a polynomial set P, we add a vertex for each variable. Then, for each polynomial, we connect all of its variables pairwise to form a clique.

Ideally, the *associated graph* of a polynomial set is chordal, to allow us to compute a perfect elimination order on it. However, this may not always be the case. Hence, if the graph is not chordal, we still want to compute a good ordering on it with the same heuristics as for chordal graph; for this reason, we will consider chordal completions.

As a convenient way of talking about chordal completions of associated graphs of polynomial sets, we define the notion of a *chordal structure* according to [LXZZ21].

Definition 2.8.2 (Chordal structure of a polynomial set ([LXZZ21], Def. 2.5))**.** *Let* $P \subseteq \mathbb{R}[x_1,\ldots,x_n]$ *be a set of multivariate polynomials. Then, a Graph* G *is a* chordal structure *of* P*, if* G *is a chordal completion of* $\mathcal{G}(P)$*. If* $\operatorname{var}(P) = \emptyset$*, any chordal graph is regarded a chordal structure of* F*.*

Li et al. have proven the following propositions regarding the preservation of the chordal structure:

- If G is a chordal structure of $\{f\} \subseteq \mathbb{R}[x_1,...,x_{n-1}][x_n]$, then G is a chordal structure of $\text{coeff}(f) \cup \{\text{cont}(f), \text{disc}(f)\}$. ([LXZZ21], Prop. 3.1)
- If G is a chordal structure of $\{f, g\} \subseteq \mathbb{R}[x_1,...,x_{n-1}][x_n]$ and G has a perfect elimination ordering with $x_n < x_i$ for $i < n$, then G is a chordal structure of $\{\text{res}(f, g)\}$. ([LXZZ21], Prop. 3.2)
- If G is a chordal structure of $A \subseteq \mathbb{R}[x_1,...,x_{n-1}][x_n]$, then G is a chordal structure of $A \cup \mathcal{F}(A)$. ([LXZZ21], Prop. 3.3)

From this, the following proposition follows immediately:

Proposition 2.8.1 ([LXZZ21], Proposition 4.1)**.** *Let $A \subseteq \mathbb{R}[x_1,...,x_{n-1}][x_n]$. Suppose that a projection operator* $\text{Proj}(A)$ *only consists of some coefficients, contents, resultants and discriminants of the polynomials in $A \cup \mathcal{F}(A)$ and G is a chordal structure of A with a perfect elimination ordering such that $x_n < x_i$ for $i < n$, then G is also a chordal structure of the polynomial set* $\text{Proj}(A)$*.*

This proposition also applies to the operators $\mathcal{F}(\text{Proj}_{br})$ and Proj_{mc} we introduced previously. By induction, it is possible to obtain the following:

Proposition 2.8.2 ([LXZZ21], Proposition 4.2)**.** *Let $A \subseteq \mathbb{R}[x_1,...,x_n]$. Suppose the graph G is a chordal structure of A and $x_n < ... < x_1$ is a perfect elimination ordering of G. If a projection operator* $\text{Proj}(S)$*, with S the polynomial set on which it operates, consists of some coefficients, contents, resultants, discriminants and some subresultants of the polynomials in $S \cup \mathcal{F}(S)$ and $(P_n = A, P_{n-1}, ..., P_1)$ is a projection procedure of A obtained via the projection ordering $x_n < ... < x_1$, then G is a chordal structure of any P_i for $1 \le i \le n$*

This theorem also implies the following:

Lemma 2.8.3. *Let $A \subseteq \mathbb{R}[x_1,...,x_n]$. Let $G = (V,E) = \mathcal{G}(A)$ be the associated graph of A. Let $F = \textsc{Elimination-Game}(G, x_n < ... < x_1)$ be the set of fill edges introduced by the elimination game. Then, $G' = (V, E \cup F)$ is a chordal structure of A and, by Proposition 4.2, also a chordal structure of the sets $(P_n = A, P_{n-1}, ..., P_1)$ obtained by the projection procedure using a projection operator as specified in proposition 2.8.2.*

Proof. If we play the Elimination Game again on G' using $x_n < ... < x_1$, no additional fill-in is introduced, so G' is chordal. Since G is a subgraph of G', G' is a chordal structure of A and we obtain the proof by proposition 2.8.2. $\square$

This shows us that the elimination game, which was introduced to simulate the fill-in for Gaussian elimination, can also give an "upper bound" for the fill edges introduced by projection in the associated graph of a polynomial set. These fill edges correspond to new polynomials, which "connect" two variables that did not previously appear inside a polynomial. For example, consider the following polynomial sets:

$$P := \{x_1 + x_4, x_2 + x_4, x_3^2 + x_2, x_3^3 + x_1, x_5 + x_2, x_5 + x_1 + x_2\} \tag{2.1}$$

$$Q := \{x_1x_2x_3, x_1 + x_3 + x_4, x_1 + x_4x_5, x_1^2 + x_5 + x_2^3\} \tag{2.2}$$

In Figure 2.2 we can see a chordal structure for P and Q respectively. Additionally, the displayed vertices together with the black edges only form the associated graphs

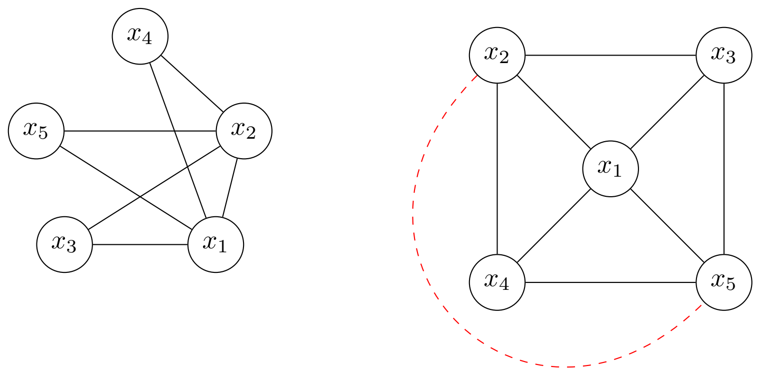


Figure 2.2: The associated graphs of P and Q

$\mathcal{G}(P)$ and $\mathcal{G}(Q)$. In the case of P, the associated graph is already a chordal structure of the polynomial set. Therefore, we can determine a perfect elimination order (for example, $x_4 < x_5 < x_3 < x_1 < x_2$). For Q, the associated graph is non-chordal, so we have to find a chordal completion together with a minimal elimination order. Here, we could use $x_4 < x_3 < x_2 < x_1 < x_5$, which would require only one fill edge (dashed, in red) to be added to the graph.

It is possible that the associated graph is not connected. This poses a minor problem for the procedure, as the elimination tree is only defined for connected graphs. Li et al. do not give a suggestion for an ordering when the associated graph is not connected, but there are different options: One that is easy to implement is simply to treat the connected components of the graph individually. We can run a minimal chordal completion to make all components chordal if necessary. Then, any elimination order on the full graph which, reduced to a single component yields a peo of that component, is a peo of the full graph. Given the nature of the projection operators, we will never compute a resultant between two polynomials from different components (no shared variable). Hence, the relative order of variables between different components has no influence on the CAD runtime and does not need to be optimized, but the individual per-component orders still matter. Another very simple option is simply to detect whether the graph is connected and use a fallback ordering otherwise.

In the general case, any finite undirected graph can be isomorphic to the associated graph of a given polynomial set. Hence, we can not make any assumptions about our graph that would allow us to use specialized algorithms with better runtime than their naive counterparts, e.g. for finding the elimination tree of minimum height.

2.8.1 Impact of the Elimination Tree

Li et al. have proven that the elimination tree for a peo can be used to obtain a better upper bound on the worst-case runtime for CAD (if variables are projected using the peo) ([LXZZ21], Theorem 5.8, 5.9). If the CAD projection is performed according to a peo, the same sets of projection polynomials can be obtained if the projection is performed “along the branches of the elimination tree”, which will be formalized below. However, the latter procedure involves a smaller number of successive projection operations on the same set, which, among other things, contributes to the better runtime bound.

Projection Along the Elimination Tree

Let $P \subseteq \mathbb{R}[x_1,...,x_n]$ be a polynomial set, $G = (V, E) = \mathcal{G}(P)$ its associated graph. We assume that $x_n < x_{n-1} < ... < x_1$ is a perfect elimination order, and $T = (V, B)$ is a corresponding elimination tree.

Now, remember that the projection sets in a classical CAD projection are defined as $P_n = P, P_i = \text{Proj}_{mc}(P_{i+1}, x_{i+1})$ for $1 \leq i < n$. If we visualize the inputs and outputs to all projection operations, we obtain a projection path that looks like a straight line. Here is an example for the set P from section 2.8, with the peo $x_5 < x_4 < x_3 < x_2 < x_1$:

$$\begin{tikzcd} P_5 = P \arrow[r, "x_5"] & P_4 \arrow[r, "x_4"] & P_3 \arrow[r, "x_3"] & P_2 \arrow[r, "x_2"] & P_1 \end{tikzcd}$$

However, we can also define a new projection procedure, that is defined as follows:

$$A_p := \{f \in P \mid \min(\text{var}(f)) = x_p\} \tag{2.3}$$

$$T_p := A_p \cup \bigcup_{\{x_l \mid (x_l, x_p) \in B\}} \text{Proj}_{mc}(T_l, x_l) \tag{2.4}$$

These sets define a projection “along the elimination tree”: For leaves x_l of the tree, the corresponding projection set $T_l := A_l$ is simply a subset of the input polynomials (such that for every polynomial, x_l is the first-eliminated variable in it). For a non-leaf node, the corresponding set can be computed by applying the projection operator to the sets associated with its children, and then forming the union of the resulting projection sets and A_l. Thus, we now obtain a “projection tree”, that is shaped exactly like the elimination tree. Again, here is the example for the set P from section 2.8:

$$\begin{tikzcd} T_5 \arrow[dr, "x_5"] & & \\ T_4 \arrow[r, "x_4"] & T_2 \arrow[r, "x_2"] & T_1 \\ T_3 \arrow[ur, "x_3"] & & \end{tikzcd}$$

As proven by Li et al., the following propositions hold for these sets ([LXZZ21], Proposition 5.6): For $1 \leq i \leq n$, we have

$$\{f \in T_i \mid x_i \in \text{var}(f)\} = \{f \in P_i \mid x_i \in \text{var}(f)\} \tag{2.5}$$

$$\{f \in \bigcup_{1 \leq i \leq n} T_i \mid \text{var}(f) \neq \emptyset\} = \{f \in \bigcup_{1 \leq i \leq n} P_i \mid \text{var}(f) \neq \emptyset\} \tag{2.6}$$

This shows us that the elimination-tree-based projection procedure can produce the same full set of projection polynomials as the original one. In particular, one could use $\{f \in T_i \mid x_i \in \mathrm{var}(f)\}$ instead of $\{f \in P_i \mid x_i \in \mathrm{var}(f)\}$ to perform the lifting step into the i-th dimension.

The projection along the elimination tree allows for a new complexity analysis, which results in a lower bound on the runtime of CAD when a peo is used to order the variables ([LXZZ21], Theorem 5.8 and 5.9). Since the projection sets are equal up to constants (eq. (2.5)), the same runtime improvements carry over to the regular CAD if the used order is a peo with a short elimination tree (relative to the number of variables).

New complexity analysis [LXZZ21]

By analyzing the projection along the elimination tree, Li et al. were able to give a better upper bound on the number of CAD cells using the method from Bradford et al. [BDE$^+$16] presented in Section 2.2.2.

It is key to observe that the doubly exponential complexity is introduced by the n-fold successive application of the projection operator on the input polynomial set. Because the projection operator has the potential to square the magnitude or the combined degree in the worst-case, repeated application will result in a "power tower" expression which can be expressed as a double exponential in n (the number of variables). Hence, by reducing the number of successive applications of the projection operator, one is able to obtain a better lower bound. First, consider the following lemma to estimate the size of the set T_p, which is the set of projection polynomials as defined in equation 2.3:

Theorem 2.8.4 ([LXZZ21])**.** *For a variable x_p, if the set A_p has the (m,d)-property and the set T_l also has this property for each $x_l \in \mathrm{child}(x_p)$, then T_p has the $\left(\left(|\,\mathrm{child}(x_p)| + 1\right) M, 2d^2\right)$-property. When $m > 1$, the set T_p has the $\left(\left(|\,\mathrm{child}(x_p)| + 1\right) m^2, 2d^2\right)$-property.*

Here, we can observe an interesting fact for our runtime: If we have a variable x_p with multiple children, then the m-value of the (m,d)-property will only increase linearly with the number of children, and d does not increase at all (if the d-value for all children can be bounded by d). If we would use a classic projection procedure, then we would have successive projections for all of the child variables again, which would lead to a double exponential growth for the m and d values. With the projection along the elimination tree however, the maximum number of successive projections is given by the height of the tree (which can be lower than the total number of variables). By repeatedly applying the lemma to get an estimate for all T_i in the tree, it is possible to obtain the following estimate:

Theorem 2.8.5 ([LXZZ21], Theorem 5.9)**.** *If the set A_l has the (m,d)-property for every x_l, then the number of CAD cells in $\mathbb{R}^n$ is at most $\prod_{i=1}^{n}(2K_i + 1)$, where*

$$K_i = \begin{cases} md, & \textit{if } x_i \textit{ is a leaf node} \\ \left(2\left(w+1\right)\right)^{2^{h_i}-1} M^{2^{h_i}-1} d^{2_i^h} & \textit{otherwise} \end{cases}$$

, where $w := \max\{|\,\mathrm{child}(x_l)| \mid 1 \leq l \leq n\}$ is the maximum number of children of any tree node and $h_i := h(x_i)$ is the height of the node x_i in the tree T.

As a simplified form, we can write $O\left((2\,(w+1))^{2^h-1}\,M^{2^h-1}d^{2^h}\right)$, where h is the height of the elimination tree. With this estimate, we should expect that the runtime of CAD can be improved by choosing an ordering that minimizes the height of the elimination tree.

Chapter 3

Heuristics for Chordality-based Ordering Methods

In this chapter, we present new ordering heuristics upon the idea of a chordality-based ordering as presented in [LXZZ21]. First, we will describe an implementation of the ordering from [LXZZ21] for the `NewCADModule` in SMT-RAT. Based on this implementation, a slight modification of the given algorithm using vertex choice heuristics and a related method which utilizes a labelling for the associated graph are proposed, both with the aim to introduce additional information about the polynomials into the algorithm.

3.0.1 Implementation of the Ordering in SMT-RAT

Li et al. proposed and evaluated their heuristic in the context of an isolated CAD algorithm. However, we are using CAD - in a slightly modified way - only as a single component in an SMT solver. Hence, there are some additional considerations we have to make for our implementation. In an SMT framework, the CAD solver is possibly called multiple times with a slightly different problem set. The incremental CAD in SMT-RAT exploits this by saving intermediate states of previous computations. If we applied a new ordering for every call to our CAD solver, the stored projection sets and sample points would be invalid and we have to compute a new CAD from scratch. It is possible that the reduction in runtime, which could possibly be achieved by choosing an ideal ordering for every run, is outweighed by the additional work for recomputing compared to the incremental CAD. SMT-RAT currently recomputes the ordering whenever a new variable is added or removed, since this requires the solver to clear the data structures anyways.

Li et al. make the following suggestions for finding a variable ordering for cylindrical algebraic decomposition[LXZZ21]:

1. "[When the associated graph of the input polynomial set is chordal], it can be better to compute the CAD via the perfect elimination orderings which result in perfect elimination tree of the minimum height" [LXZZ21]

2. “[When the associated graph of the polynomial set is nonchordal], but still nearly chordal (and also sparse), it could be better to use the variable orderings that result in minimal chordal completions of the original system” [LXZZ21]

We implement these suggestions in section 3.0.1. The source code of the C++ implementation used in SMT-RAT can be found at [(kr23], in `src/smtrat-cad/variableordering/chordal_vargraph_elimination_ordering.cpp`. The presented algorithms uses two subalgorithms that are not specified: isConnected checks whether the graph is connected and returns a boolean value accordingly. In the C++ implementation, we use the Boost Graph Library to implement this functionality. VOFallback is a fall-back variable ordering that can be configured in the settings of SMT-RAT.

```
procedure Chordal($P \subseteq \mathbb{R}[x_1,...,x_n]$)
    $G \leftarrow (V \leftarrow \emptyset, E \leftarrow \emptyset))$
    for all polynomials $p \in P$ do
        $V \leftarrow V \cup \text{var}(p)$
        for all pairs of different variables $\{x, y\} \subseteq V$ do
            $E \leftarrow Ecup\,\{x, y\}$
            $d(x, y) \leftarrow \max\{d(x,y), \deg(p,x)\}$
            $d(y, x) \leftarrow \max\{d(y,x), \deg(p,y)\}$
        end for
    end for                                        ▷ We have constructed $G \leftarrow \mathcal{G}(P)$
    if not isConnected($G$) then
        return VOFallback(P)
    end if
    $\alpha, F \leftarrow$ MCS($G$)
    if $F = \emptyset$ then                                                  ▷ $G$ is chordal
        $(\alpha, \_) \leftarrow$ E-Tree($G$) ▷ The returned tree is ignored, only the peo is stored
                              ▷ $\alpha$ is a peo that generates a minimal elimination tree
        return $\alpha$
    else
        $\alpha, F \leftarrow$ MCS-M($G$)                 ▷ $\alpha$ is a minimal elimination order (meo)
        $G^+ \leftarrow (V, E \cup F)$
        $(\alpha, \_) \leftarrow$ E-Tree($G$)
        return $\alpha$
    end if
end procedure
```

A variable ordering in SMT-RAT is defined by a function, which takes a set P of polynomials and computes a peo in the form of a variable sequence, which is a permutation of the variable set var(P). The variable ordering function to use is configured as part of the settings of the `NewCADModule`.

First, we construct the associated graph of the polynomial. Then, we check whether the graph is connected, since the result of E-tree is only defined for connected input graphs. If the graph is non-connected, an unspecified fall-back ordering is used. In the C++ implementation, this is a template parameter which can be set to any other available ordering in SMT-RAT, like the triangular ordering. If the graph is connected, we check for chordality by running the MCS algorithm and then checking whether the generated elimination order is perfect by calling Elimination-Game.

If the graph is chordal, we call E-Tree(G) to obtain the elimination tree of minimum height and a new peo which generates that elimination tree. This peo is then returned by the algorithm.

If the graph is not chordal, we compute a minimal elimination ordering together with the set of fill edges F using MCS-M. These fill edges are then added to the graph (making it chordal), and E-Tree is called. The resulting fill edge count is a simple indicator for how close the graph is to being chordal. If the fill-in is very large, then other orderings that do not consider chordality might result in similar fill-in, while being more optimal w.r.t some other criterion. In these cases, it may be more sensible to choose another ordering over the chordality-based ordering. However, such a case-distinction has not yet been implemented and tested in the algorithm. The set of fill edges is then added to E to obtain the chordal graph G^+. In the next step, E-Tree(G) is called, which returns an elimination tree of minimum height for G^+, together with a peo on the filled graph that produces this tree. The elimination tree itself is not needed and therefore not stored in the pseudocode. The returned peo on the filled graph possibly has a shorter elimination tree than the original meo returned by MCS-M (which is also a peo on the filled graph). Considering all chordal completions of the input graph, it is possible that there is a peo with an even shorter elimination tree than the one obtained by our procedure. However, as shown by Pothen [Pot88], finding the elimination tree of minimum height among all peo for all chordal completions of a graph is an NP-complete problem. For this reason, we consider only a single minimal chordal completion to obtain an approximation to the elimination tree of minimum height.

It is also possible to simplify the algorithm by only running MCS-M to detect chordality (by absence of filledges) and compute a minimal fill immediately. However, the $O(|V| \cdot |E|)$ complexity of MCS-M is slightly worse than MCS, which runs in $O(|V| + |E|)$.

3.1 Better Integration of Polynomial Properties

The idea of modelling variable elimination using the elimination game in graphs was first explored in [Par61] for systems of linear equations. To construct the associated graph from a matrix $A \in \mathbb{R}^{n \times n}$, an edge x_i, x_j is added to the graph iff either a_ij or a_ji is nonzero. Therefore, a pair of vertices x_i, x_j with $i \neq j$ always corresponds to two entries in the matrix. We lose information about the exact value of the coefficients, but in most computer architectures, fixed-width numbers are used - so the runtime of a single arithmetic operation is mostly independent from the value. In addition, Parter indicates that the elimination game simulates the Gaussian elimination accurately - in the sense that "almost always", the associated graph of the matrix after an elimination step is equal to the result of the elimination game applied to the associated graph of the original matrix. To apply the same idea to systems of polynomial constraints, Li et al. use a very similar definition for the associated graph: It is constructed as an undirected, unlabelled graph using the set of variables as the vertex set, where two variables are connected via an edge iff they appear together in a polynomial. However, sets of polynomials have more structural complexity than systems of linear equations, so we lose more information in the abstraction. First, we lose all information about the degrees of the variables in the polynomials. In the lifting phase of the CAD, the number of sample points that a single polynomial contributes when lifting a single point is given by the number of zeroes, which is bounded by the degree. Therefore, the degree has a major impact on the complexity of CAD, which is also shown in the complexity analysis presented in section 2.2.2. This has been recognized in previous research on variable ordering, so most heuristics prefer to eliminate variables that appear to low degree in the polynomial set. This is visible in the triangular ordering (see definition 2.7.1), which orders the variables based on their maximum degree in the polynomial set, using other degree-based criteria for tie-breaking. The maximum-degree criterion even ignores most other aspects of the variable, such as the number of terms / polynomials it occurs in and the other variables it appears together with.

Furthermore, any set of variables can appear in an arbitrary number of polynomials, which means that an edge in the associated graph for a polynomial set could also represent an arbitrary number of polynomials. The number of adjacent vertices can be weakly correlated to the number of terms the variable appears in, especially if the average number of variables in a single term is small. In addition, a variable appearing in a large number of different polynomials is more likely to be adjacent to two vertices that are not connected, which would mean that it cannot be eliminated without introducing fill. The size of the projection sets can have a large influence on the runtime of the CAD, in particular the first sets: Through the calculation of resultants in Proj_{mc}, the size of a polynomial set can grow quadratically through projection. In the lifting phase, a single point is lifted with every polynomial of the current level, thus the number of lifting points grows with the number of polynomials.

For these reasons, we will now propose methods to include this information into chordality-based ordering heuristics, with the hope that this allows the solver to select a better variable ordering.

3.2 Choosing Better Vertices in Graph Algorithms

In section 3.0.1 we introduced an implementation of the chordality-based heuristic proposed by Li et al. for CAD. As we have seen, there are two subalgorithms that are used to determine elimination orderings: E-Tree (algorithm 4), which is used to determine a peo for chordal graphs, and MCS-M (algorithm 3), which is used to determine an meo for non-chordal graphs. In both algorithms, there is a point where a vertex is arbitrarily chosen from a set, these points are underlined in the listings. At these points, we can utilize a secondary ordering on the variables to choose a vertex whose variable is minimal w.r.t the ordering. This allows us to take additional information about the variable into account, while still producing a meo or a minimal elimination tree respectively. The orderings presented in section 2.7 are good choices for secondary orderings, since they make their decisions mostly based on the degree of the variables, which is completely lost in the associated graph.

For the evaluation, it can be interesting to know how much influcence the secondary ordering had during a given run. Therefore, we calculate a "degree of re-ordering" as follows: First, a "choice ratio" is computed at every execution of the underlined step as the number of vertices that can be chosen, divided by the total number of vertices that have not been eliminated already. For E-Tree, as given in the pseudocode, the value is simply $\frac{|U|}{|V|}$. For MCS-M, the chosen vertex has to fulfill the maximum-weight condition, so the choice ratio can be written as $\frac{|\{v \in V \setminus \alpha | \forall v' \in V \setminus \alpha . w(v) \geq w(v')\}}{|V \setminus \alpha|}$. The degree of re-ordering is computed as the product of the choice ratio of every iteration. If we make the simplifying assumption that the choice of a vertex does not influence the choice ratios in the following iterations, the degree of re-ordering expresses the ratio of the possible elimination orderings that can be achieved using the algorithm, divided by the total number of orderings, $|V|!$. Thus, a value of 1 would represent the case that every possible ordering can be generated by the graph, which (for both algorithms) is the case when the graph is a clique. While it is wrong to assume that the choice of a vertex does not influence the following choices, we still expect the degree of reordering to be a useful approximation of the influence of the secondary ordering.

The C++ implementation of section 3.0.1 allows to specify a secondary ordering as a template parameter, which is used as the secondary ordering for the aforementioned graph algorithms. To realize the basic implementation as specified in [LXZZ21], a pseudo-random variable ordering is used as the secondary.

3.3 Extending the Graph Representation

To obtain an even tighter integration of polynomial properties into a chordality-based algorithm, the associated graph itself can be labelled. To find a peo on the resulting graph, we can use the very simple peo-finding algorithm proposed in section 2.3, based on the repeated removal of simplical vertices. In each step, a vertex can be chosen that minimizes some property that is computed using the labels. For example, we could label each edge with the number of polynomials contributing to it and choose the vertex for which the sum of labels of incident edges is minimal.

However, the accurate representation of the elimination process in the labels poses a challenge, since we want the labels after the vertex elimination to be accurate for the polynomial set after the projection. This has some implications on our labelling process: First, we might have to recompute labels during the elimination process to match the projection polynomial set. Second, the labelling has to be precise enough to contain all information that is needed to re-compute the labelling accurately.

For example, we could try to label the edges with the number of polynomials as suggested above. This labelling cannot be recomputed accuratly, since we don't know whether two edges belong to the same or different polynomials. A possible way to address this is a labelling with sets of identifiers, where each edge is labelled with the set of polynomials that produce it. Now, we could compare the sets during elimination, and possibly introduce new identifiers for the newly generated polynomials. However, with this kind of labelling, we have to deal with the problem of possibly quadratic growth during elimination, which would result in the same doubly exponential bound for the worst-case runtime as the full CAD procedure. To avoid this problem altogether, it is often easier to define the labels themselves as upper bounds of some properties of the associated polynomials (here: size). This way, we can assume the worst case for the elimination, to compute accurate upper bounds for the new polynomials as labels.

In the following, we will present such a labelling system with the associated algorithm based on the maximum degree of a variable. In order to analyze this labelling, we first define $\delta(P, x, y) := \max\{\deg(p, x) \mid \{x, y\} \in P\}$ as a shorthand for the maximum degree of x among all polynomials in p that contain x and y. Given our associated graph $\mathcal{G}(P) = (V, E)$, we will label both sides of every edge using a mapping $d : V \times V \to \mathbb{N}$. For a given edge $\{v, w\} \in E$, $d(v,w)$ gives the label on the v-side of the edge, while $d(w,v)$ gives the label on the w-side. When $\{v,w\} \notin E$, we define

$d(v,w) = d(w,v) = 0$. The core idea of the labelling can be described as follows: The label $d(v,w)$ in the associated graph $\mathcal{G}(P)$ should approximate $\delta(P, v, w)$.

In particular, we want to update the labels during elimination in order to keep the approximation accurate for $\text{Proj}_{mc}(P)$. To update the edge labels during elimination of v, we consider each pair of vertices $x, y \in \text{adj}(v)$ that are adjacent to v and update the labels for $\{x, y\} \in E$ as follows (Note: The edge is created per definition of Elimination-Game):

$$d(x,y) \leftarrow \max\{d(x,y), (d(v,x) + d(v,y)) \cdot d(x,v)\} \tag{3.1}$$
$$d(y,x) \leftarrow \max\{d(y,x), (d(v,x) + d(v,y)) \cdot d(y,v)\} \tag{3.2}$$

With this update, we want to simulate the calculation of the resultants, which have the most impact on the degree among all operations in Proj_{mc}: We assume that there is $p \in P$ with $\{v, x\} \subseteq \text{var}(p)$ and $q \in P$ with $\{v, y\} \subseteq \text{var}(q)$. Through $d(v, x)$ and $d(v,y)$, we obtain an approximation of the degree of v in p and q respectively. The dimension k of the Sylvester matrix used in the resultant calculation is defined as $\deg(p[x]) + \deg(p[y])$, so $d(v,x) + d(v,y)$ is an upper bound for this dimension. In the worst case, we can assume that x appears as $x^{d(x,v)}$ and y appears as $y^{d(y,v)}$ in the coefficients of $p[v]$ and $q[v]$. The resultant is computed as the determinant of the sylvester matrix, which in turn is computed as the sum of products of k entries in the matrix. In the worst case, such a summand can be of the form $a \cdot \prod_{i=1}^{k} x^{d(x,v)} y^{d(y,v)} = x^{k \cdot d(x,v)} y^{k \cdot d(y,v)}$, where a is a term not containing x or y. By substituting k, we arrive at the upper bound $(d(v,x) + d(v,y)) \cdot d(x, v)$ for the degree of x in the resultant and $(d(v,x) + d(v,y)) \cdot d(y, v)$ for y in the resultant.

It should be noted that there is an edge case that was not considered in this procedure for computing the upper bound: When p is a polynomial containing v but neither x or y, and q is a polynomial containing $\{x, y\}$ but not v, the sylvester matrix (and thus the resultant) is still defined. In this case, it would take the form of a diagonal matrix of dimension $\deg(p, v)$ with q on the diagonals, which means that $\deg(r, x) \leq \deg(p, v) \cdot \deg(q, x)$ and $\deg(r, y) \leq \deg(p, v) \cdot \deg(q, y)$. The effect of this resultant was only noticed after the evaluation with the new ordering was already performed, so it is not taken into account for the update computation. For this reason, the $d(x,y)$ are not proper upper bounds, but only approximations.

Now that we have defined an accurate labelling based on the maximum degree, we can use it in a chordality-based heuristic to greedily choose a vertex that results in an optimal elimination in the current graph. The algorithm, which we refer to as Greedy, Degree-minimal Chordal Completion (GDCC), is given in algorithm 5.

Similar to section 3.0.1, the algorithm starts by computing the associated graph. While the graph is computed, the edge labels $d(x,y)$ are computed using the input polynomials.

Next, two weight maps w_m and w_d are initialized and then populated for all vertices still in the graph. w_m maps a vertex to the number of fill edges that would be introduced by eliminating it. w_d is a little more complicated: For a given variable v, $w_d(v)$ is computed as the sum of the new values for $d(x,y), d(y,x)$ *after the elimination* for all pairs $x \neq y$ of variables that are adjacent to v. This value acts as a rough approximation to the sum of degrees of the polynomials added through Proj_{mc}. It is, of course, not exact, especially since we are missing information on the number of polynomials associated to an edge - but it should give us an approximation that is close enough for heuristic purposes.

After the values for w_d and w_m are computed, we select a minimal vertex w.r.t $<^w$. The operator $<^w$ is a comparison operator on the vertices using information from w. Two variants of the operator are currently defined in the algorithm:

$$u <^*_{md} v :\Leftrightarrow w_m(u) < w_m(v) \vee (w_m(u) = w_m(v) \wedge w_d(u) < w_d(v)) \tag{3.3}$$

$$u <^*_{d} v :\Leftrightarrow w_d(u) < w_d(v) \tag{3.4}$$

The variant $<^w_d$ only compares the w_d-values in the graph. In contrast, $<^w_{md}$ orders by the number of added fill-edges first, and then uses the degree-based ordering as a tie-breaker. This ensures that if a simplical vertex exists, it is chosen for elimination. As described in section 2.3, the algorithm with using $<^w_{md}$ will always return a peo if the input graph is chordal. Therefore, we will refer to this as the standard implementation of GDCC. Since the restriction to simplical vertices might require a suboptimal choice w.r.t our degree metric w_d, we also implement $<^w_d$ to compare it later in the evaluation.

When a vertex v was chosen, it is eliminated and appended to the variable ordering α. Edge labels are recomputed using eq. (3.1). The steps of populating w_d, w_m, choosing a minimal vertex w.r.t $<^w$ and eliminating this vertex are repeated until V is empty. At this point, the ordering α is returned.

Algorithm 5 GDCC

```
procedure GDCC($P \subseteq \mathbb{R}^n$)
    $G \leftarrow (V \leftarrow \emptyset, E \leftarrow \emptyset))$
    $d : V \times V \to \mathbb{N} \leftarrow 0$          ▷ A two-sided edge label as described above.
                                                          ▷ All possible values are initialized to zero
    for all polynomials $p \in P$ do
        $V \leftarrow V \cup \mathrm{var}(p)$
        for all pairs of different variables $\{x, y\} \subseteq V$ do
            $E \leftarrow E \cup \{x, y\}$
            $d(x, y) \leftarrow \max\{d(x,y), \deg(p,x)\}$
            $d(y, x) \leftarrow \max\{d(y,x), \deg(p,y)\}$
        end for
    end for
    $\alpha = ()$
    while $V \neq \emptyset$ do
        $w_m : V \to \mathbb{N} \leftarrow 0$          ▷ Initialize the weight maps in every iteration
        $w_d : V \to \mathbb{N} \leftarrow 0$
        for all $v \in V$ do
            for all pairs $\{x, y\} \subseteq \mathrm{adj}(v)$ do
                if $\{x,y\} \notin E$ then
                    $w_m(v) \leftarrow w_m(v) + 1$
                end if
                $w_d(v) \leftarrow w_d(v) + (d(v,x) + d(v,y)) \cdot d(x, v)$
                $w_d(v) \leftarrow w_d(v) + (d(v,x) + d(v,y)) \cdot d(y, v)$
            end for
        end for
        $v \leftarrow \min_{<^w}(V)$                     ▷ Find minimal vertex w.r.t $<^w$
                                                          ▷ The ordering is defined by the weight maps
        $\alpha \leftarrow \alpha \circ (v)$                     ▷ $v$ is appended to the peo
        for all pairs $\{x, y\} \subseteq \mathrm{adj}(v)$ do
            $d(x,y) \leftarrow \max\{d(x,y), (d(v,x) + d(v,y)) \cdot d(x, v)\}$
            $d(y,x) \leftarrow \max\{d(y,x), (d(v,x) + d(v,y)) \cdot d(y, v)\}$
        end for
        $E \leftarrow E \setminus \mathrm{adj}(v)$
        $V \leftarrow V \setminus \{v\}$
    end while
    return $\alpha$
end procedure
```

Chapter 4

Experiments

The goal of this thesis is an evaluation of the presented variable ordering heuristics in the context of SMT solving. To evaluate this, we will be comparing the following variable ordering heuristics implemented in SMT-RAT:

- The Triangular Ordering, specified in `smtrat::NewCADSettingsTriangular`
- A pseudo-random ordering, which works by seeding a PRNG using properties of the constraints, and then shuffling the variables using random numbers produced by the PRNG. This is specified using `smtrat::NewCADSettingsPseudorandom`.
- The chordality-based ordering from section 3.0.1, specified using `smtrat::NewCADSettingsChordal`
- The chordality-based ordering from section 3.0.1, with the elimination tree minimization step disabled, specified in `NewCADSettingsChordalNoETree`
- The chordality-based ordering from section 3.0.1, using the triangular ordering to choose vertices as described in section 3.2. This ordering is specified using `smtrat::NewCADSettingsChordalTriangular`
- The GDCC algorithm, as proposed in section 3.3, specified using `smtrat::NewCADSettingsDMFillChordal`
- A variant of GDCC, which uses the operator $<^w_d$ instead of $<^w_{md}$. It is not guaranteed to compute a peo when one exists, because it only considers the degree criterion. It is specified using `smtrat::NewCADSettingsDMFill`

For all orderings, the following common settings, implemented as `smtrat::NewCADBaseVariableOrderingSettings`, are used for the `NewCADModule`:

- Use Proj_{mc} as the projection operator
- Apply full incrementality and allow for ordered backtracking, as described in [KA20]

All settings listed above are implemented in the file `src/smtrat-modules/NewCADModule/NewCADSettings.h.in`

The `NewCADModule` configured this way is then used as a single theory backend to the boolen SAT solver component of SMT-RAT. The settings for the NewCADModule and the strategy are defined in A.1 and A.2 respectively.

To gain a deeper insight into the properties of the CAD execution and variable ordering, the code logs various information into a statistics module. For the chordality-based ordering, this includes the number of edges, vertices, fill edges and the height of the generated elimination tree. While only the chordality-based ordering (and, to some extent, GDCC) use the graph structures to derive heuristic choices, it is possible to define the associated graph and the elimination tree for any ordering. As this data can be useful for comparing heuristics, we introduced an analyzer that runs after any ordering which is not derived from the chordal ordering to compute these values: First, it generates the associated graph of the input polynomial set identically to the chordal ordering. Afterwards, ELIMINATION-GAME is called with the associated graph and the derived ordering to compute the number of fill edges, which is stored into the statistics module. Then, an elimination tree is generated for the given ordering, which is now a peo on the filled graph. Since no minimization of the elimination tree height should be done and the order should not be changed, the tree is generated using an algorithm based on definition 2.3.10 instead of using E-TREE. The height of this elimination tree is then computed and stored into the statistics as well.

To test each individual strategy, the `__CADSETTINGS__` placeholder in A.2 is replaced with a corresponding settings struct, inheriting from `smtrat::NewCADBaseVariableOrderingSettings` and specifying the variable ordering to be tested. After replacing the placeholder value, the solver is built by running CMake with the command line `cmake -D CMAKE_BUILD_TYPE=RELWITHDEBINFO -D SMTRAT_Strategy=CADVOTest -U LOGGING -D SMTRAT_DEVOPTION_Statistics=ON ...` This enables release-level compiler optimization while still embedding debug info in the executable, specifies our test strategy, explicitly disables logging and enables the builtin statistics collection. The solver is then built using the target `smtrat-shared`.

The solver is built from commit `1c67006c8baf56b5f6dc344771db8f80bda5ffba` of the repo accessible at [(kr23]. The CArL library, which is a required dependency and used for all polynomial operations in SMT-RAT, is built from commit `11297c86bdcf50aab93e7140029eb398f18cea96`, using the default build options. The solver is compiled using `gcc 11.3.0` on a Ubuntu 22.04 system.

To evaluate the impact of the variable orderings in an SMT-solving context, seven variants of SMT-RAT, one for each of the listed variable orderings, were built as described. Every variant of the solver was then executed on every problem of the full QF_NRA problem set from [smt], commit `4a059777fb38d24e182f468bd5a6b15f93899be0`. For each problem, the solver was executed with a 2GB memory limit and a one minute time limit using the `SlurmBackend` of `benchmax`, which is integrated into SMT-RAT. Afterwards, the problems that were not answered (with `SAT`, `UNSAT` or `UNKNOWN`) by all solvers were collected to run all solvers on them a second time, with a two minute timeout. The benchmarks were executed on a cluster of computers with 2x Intel Xeon Platinum 8160 Processors running at 2.1 GHz (Hyperthreading disabled) and 192GB RAM.

Chapter 5

Evaluation

5.1 Conventions

In the experimental data, the solvers are named after the settings struct used to specify them, without the `smtrat::NewCADSettings` part. For example, the Chordal ordering using Triangular ordering as a vertex choice heuristic, is named Chordal-Triangular in the table. When necessary, the abbreviations Cho. for Chordal, Tri. for Triangular and P.Rand. for Pseudorandom are used.

A result is counted as `TIMEOUT` resp `MEMOUT` when the solver exceeded its time or memory limit and was forcefully stopped. `SAT` and `UNSAT` are returned when the solver proved that the input problem was satisfiable resp. unsatisfiable. The solver returns `UNKNOWN` when it cannot determine the result. When counting the number of results, we will often use `SOLVED` for the sum of `SAT` and `UNSAT` results, and `ANSWERED` for the sum of `SAT`, `UNSAT` and `UNKNOWN` results.

5.2 Caveats

In the case of the strategy used here, the `NewCADModule` will return `UNKNOWN` when it detect that the CAD is incomplete (see 2.2.1). In this case, the `SATModule` can not assume `SAT` or `UNSAT` for the set of constraints it passed to the `NewCADModule`. It may try to explore different branches of the decision tree, which can possibly result in a satisfying assignment (so the solver returns `SAT`). Otherwise, the solver is forced to return `UNKNOWN`, since it cannot deduce unsatisfiability without knowing the answer to the failed CAD call.

It is important to consider that the `UNKNOWN` answers will impact the accuracy of our analysis: Since different variable orderings will result in different projection polynomial sets, it is possible that for the same input problem, `NewCADModule` returns a definite answer (or runs out of memory / time during the computation) for one ordering, while another ordering results in a problematic polynomial that forces the solver to return `UNKNOWN`. The `NewCADModule` can possibly detect incompleteness before computing the majority of a rather large CAD problem, and the SAT solver is able to prove satisfiability with an easier sub-problem, which would result in a faster answer for that particular problem. However, the opposite can happen as well, where the `NewCADModule` is unable to compute an easy CAD that would immediately prove satisfiability for the whole problem, forcing the solver to explore more of the boolean structure, possibly leading to harder CAD calls. Therefore, it is hard to judge which variable ordering performs better when both return a large number of `UNKNOWN` results.

This also implies that two solver variants which use different variable orderings might can end up solving different CAD problems when invoked on the same input problem - even with a fully deterministic SAT component, we can only guarantee that the first call will be the same. For this reason, any analysis of individual CAD calls must be done with caution when not limited to the first call for a problem, which could not be meaningful.

We must also consider that SMT-RAT, in the tested configuration, is a less-lazy SMT solver using incremental CAD. It will frequently call the theory solver to check consistency of the constraint set that corresponds to the current assignment - not only when a full assignment was found, but possibly at earlier points (e.g. before a decision is made). Therefore, it is likely that many of the first CAD calls are small subsets of the full constraint set, which can be easy compared to the later problems. This is confirmed by the data in table B.2. First, we can observe that all of the listed metrics for the difficulty of the polynomial set (the size, maximum degree, combined degree) tend to increase with later CAD calls. This is the expected effect of less-lazy SMT solving with the incremental addition of constraints. Furthermore, it is notable that the average time for the CAD call actually tends to decrease: The first call takes the largest amount of time, even though it is done on the "easiest" polynomial set on average; meanwhile, successive calls can be completed a lot quicker. This is also expected and can be explained by the incrementality features of the `NewCADModule`, which avoid a re-computation of the full CAD if only a small number of constraints was added. As a result, we can not reliably predict the performance of the full SMT solver by analyzing invididual CAD calls.

5.3 Overview

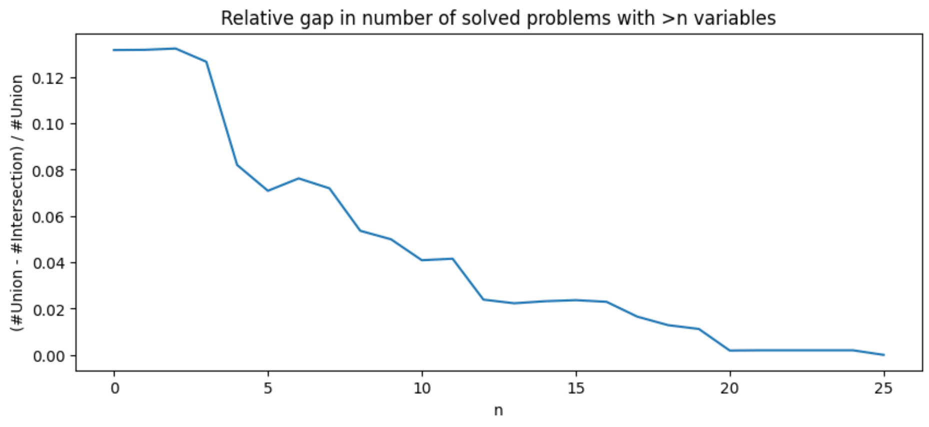
Table B.1 shows the distribution of solver results with the different variable ordering heuristics. Note that the Intersection and Union heuristics are virtual strategies to simulate a solver that can solve the Intersection resp. Union of the solvable problem sets for each solver, thereby approximating the optimal resp. least optimal combination of the strategies. Their definition can be taken from algorithms 6 and 7.

Looking at these results, we can observe that the overall performance of Chordal and its direct variations is rather poor, especially when compared with the Triangular strategy. At first, this result seems unexpected, in particular when we compare our results with those obtained by Li et al. ([LXZZ21], Table 6): In one of their experiments, they compared the chordal ordering against the variable ordering returned by the Maple command `SuggestVariableOrdering`, which uses the Triangular ordering internally. On the three test problems, the chordal ordering always performed better. All of these problems are specific instances of the polynomial set:

$$I^{n_1,n_2} = \bigcup_{0\leq i<n_1, 0\leq j<n_2} \left\{ \begin{array}{c} U_{i,j}R_{i,j+1} - R_{i,j}U_{i+1,j}, \\ D_{i,j+1}R_{i,j} - R_{i,j+1}D_{i+1,j+1}, \\ D_{i+1,j+1}L_{i+1,j} - L_{i+1,j+1}D_{i,j+1}, \\ U_{i+1,j}L_{i+1,j+1} - L_{i+1,j}U_{i,j} \end{array} \right\}$$

In particular, the tested instances were $I^{1,1}, I^{2,1}, I^{1,2}$. In this polynomial set, every variable appears at most with degree 1 in every polynomial, that means, it appears only once. In addition, most variables will appear in an equal number of polynomials for the instances cited here. Because of this structure, the variables are equal with respect to the first two criteria of the triangular ordering, and most are equal with respect to the third. In contrast, due to the large number of variables and the simple structure in the variable degrees, the associated graph can capture the structure of the set well. However, the same can not be expected from the polynomials in the QF_NRA benchmark set: First, we can observe that a large number of problems (6129/12134) contain three or less real valued variables in total. For these problems, the information captured by the associated graph is very small: In particular, there are only four graphs with less than four vertices (up to isomorphism), of which only two are connected: A single isolated vertex, two vertices connected by an edge, three vertices connected in a line and a triangle consisting of three vertices. All of these are chordal, and on all graphs except for the 3-vertex line, every ordering is a perfect elimination ordering. For such problems, the information about the number of variable occurences and their degree is more important, which is reflected by the fact that the Triangular and DMFill orderings perform best on these problems. When we consider problems with more variables, the picture starts to shift a bit: On the problem subset with at least four variables, the chordality-based strategies fare better overall. Not only is the Chordal strategy now noticeably better than Pseudorandom, we can now see ChordalTriangular performs even better than Triangular on these problems. However, it is also visible that the number of UNKNOWN results is inversely correlated to the number of solved instances. If we consider the number of answered problems, for which we count SAT, UNSAT and UNKNOWN results, we can see that this number is very close for all solvers except Pseudorandom and the virtual Intersection and Union, it is ranging from 2518 to 2524. If we assume that the solver would have terminated if the CAD was complete, this would indicate that the impact of the variable ordering on the runtime is rather small.

To analyze this effect, we consider the set of problems with more than n variables for a varying n. In section 5.3, we plot the “relative improvement” of Union over Intersection, calculated as the difference in solved instances divided by the number of instances solved by Intersection, for $0 \leq n \leq 25$.



We can notice that the relative influence of the variable ordering diminishes for higher n. In fact, for $n = 25$, the number of solved instances by Intersection and Union is equal (459 solved of 3208 total problems). In this case, all solvers manage to solve the exact same set of problems, so none of the variable orderings can give a different result than a pseudorandom order. This result is unexpected. As the total number of possible variable orderings ($n!$) increases exponentially, we would expect to see a growing impact with higher n, since there are simply more options for the resulting CAD projection sets. However, when the number or degree of polynomials is low in relation to the variable ordering, it is more likely that the polynomial set is more "symmetric" with respect to the variables, in the sense that their number or degree of occurrence is very similar. In addition, the effect that we see could also be due to the structure of the SMT-LIB benchmark set - it might be that all of the unsolved problems for $n > 25$ are significantly harder than all of the 495 solved problems, possibly because they belong to different problem categories.

5.3.1 Comparing Chordal and Nonchordal Problems

For the chordality-based orderings, it is of particular interest to see whether the performance is significantly different between chordal and non-chordal associated graphs. This would suggest that a chordality test on the associated graph could be used as a "meta-heuristic" to select the variable ordering strategy. To analyze this, we will consider the first CAD call for every problem, and only focus on problems where that call is made with at least four variables. The reason for the restriction to the first call is two-fold: In addition to the inconsistencies between later calls introduced through UNKNOWN answers, there was no problem for which SMT-RAT reset the CAD structure during solving. This means that in all cases, the variable ordering was determined once at the start. Since we gather most of the CAD-related statistics, including the variable count and the properties of the associated graph, only when the ordering is executed, we do not have these statistics for any but the first CAD call of a problem.

In table B.5 we can see the answer distributions for the individual CAD call, evaluated in a similar manner as in table B.1. While analyzing the tables, it must be taken into consideration that the total problem count varies slightly between the solvers. This happens when the first CAD run for the problems has not been started when the solver terminates, and no statistics for the CAD run have been written. In addition, the problems which result in disconnected associated graphs are not taken into account here, since the Chordal ordering would not be executed and therefore no attempt is made to find a peo. Since Chordal would fall back to using the Triangular ordering anyways, this data has to be excluded. We note that in total, 7522 of 10214 graphs generated by the Chordal ordering were connected. This can be split into 4133 of 5585 graphs with less than four vertices, and 3389 of 4629 graphs with at least four vertices. These numbers are higher than expected, so we must consider that a significant fraction of the Chordal and DMFill calls actually used the Triangular strategy. One possible cause is again the less-lazy nature of the SMT solver, which results in the first CAD call only containing a small subset of all constraints in the problem.

For the chordal graphs, we can observe that the performance of the pure Chordal strategy is similar to a pseudo-random ordering. However, we are also able to observe that the ChordalTriangular strategy performs a little better than the Triangular strategy, which is consistent with our observations for the complete SMT solving

subset	Cho.	Cho.NoETree	Cho.Tri.	DMFill	DMFillCho.	P.Rand.	Tri.
all	0.833	0.939	0.834	0.891	0.901	0.937	0.897
#vars ≤ 3	0.876	0.985	0.876	0.957	0.957	0.957	0.919
#vars > 3	0.750	0.849	0.751	0.761	0.791	0.896	0.853

Table 5.1: Average values of the elimination tree height

results. For the nonchordal graphs, the results are particularly interesting: First, we can observe that the ration of `UNKNOWN` answers to `SAT` and `UNSAT` answers is significantly higher for all solvers compared to the chordal graphs. We observe that TRIANGULAR solves the most problems here, though we refrain from further analysis of the CAD performance because of the high number of `UNKNOWN` results.

Even for strategies like PSEUDORANDOM, which do not consider chordality in any way, we note that the ratio of `UNKNOWN` results to the total problem count seems highly correlated to the chordality of the graph: In total, PSEUDORANDOM answers 1779 / 4518 (39%) of problems with `UNKNOWN`. For the chordal subset, it is only 186 / 1393 (13%) of problems while in the nonchordal subset, 1674 / 1859 (90%) of problems are answered with `UNKNOWN`. This leads to the hypothesis that in general, it is more likely for an incomplete CAD to occur if the associated graph of the input polynomial set is nonchordal. It could be interesting to investigate this in more detail, but our experiment setup does not yield any more meaningful results.

5.3.2 The Impact of the Elimination Tree

To evaluate the impact of the elimination tree height, we tested two variations on the chordal ordering from section 3.0.1: CHORDAL, which tries to order the vertices to achieve a minimum-height elimination tree, as proposed by Li et al., and CHORDAL-NOETREE that uses the same ordering strategy listed as section 3.0.1, with the only difference being that the call to the E-TREE algorithm is skipped. Therefore, both CHORDAL and CHORDALNOETREE will use a perfect elimination ordering when the graph is chordal, but only one tries to optimize the e-tree height.

In section 5.3.2, some statistics on the relative height of the elimination tree (computed as height of the tree divided by the vertex count) are listed. We can observe that the Jess and Kees method employed in our implementation has the desired effect: The CHORDAL ordering with the call to E-TREE manages to generate the shortest tree on average among all solvers in both partitions of the problem set. In particular, the height is significantly lower than for CHORDALNOETREE. When we compare these average heights with the values from table B.4, we can not establish any significant correlation. For example, we can observe that CHORDAL and CHORDALNOETREE solve almost the same number of instances, even though the average tree height is different. In contrast, CHORDALTRIANGULAR, which generates trees of almost identical height, performs better than CHORDAL due to the vertex choice heuristic. Hence, we conclude that the height of the elimination tree does not have a meaningful impact on the performance of the `NewCADModule` on the problem set at hand.

5.3.3 Degree of Reordering

In order to assess the influence of the Triangular vertex choice heuristic in the ChordalTriangular variable ordering, the degree of reordering, as defined in section 3.2 was recorded and evaluated. The results are summarized in fig. 5.1

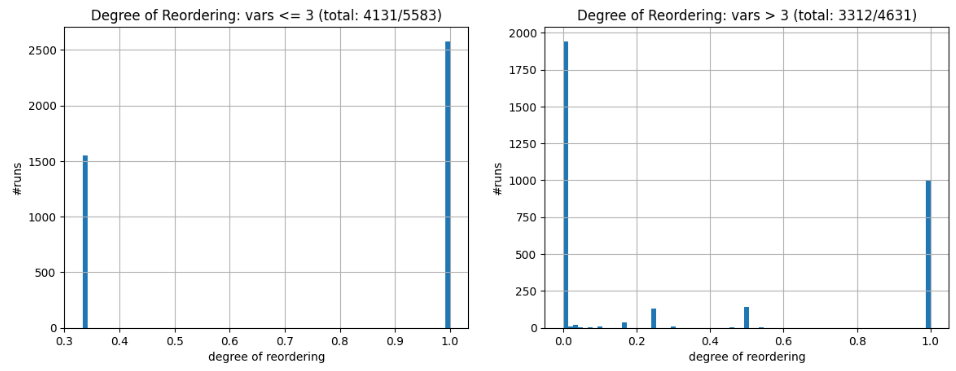


Figure 5.1: The histograms show the distribution of the degree of reordering, recorded from Elimination-Tree (algorithm 4), for all problems from the benchmark set divided into two subsets. The left histogram only shows the problems with less than three variables, while the right histogram shows the problems with at least four variables. In the diagram title, the left number shows the total number of problems for which the degree of reordering was recorded, while the number after the / shows the total number of problems in that set. Since ChordalTriangular aborts on non-connected graphs, these numbers are not equal.

For the problems with less than three variables, the choice ratio takes on only two possible values of $\frac{1}{3}$ and 1. These correspond to the expected ratios for a line graph resp. a triangle: For the line graph, the algorithm is first allowed to choose one of the ends of the line, so the choice ratio is $\frac{2}{3}$. In the next step, only two vertices remain, but we are not allowed to choose the one which was adjacent to the just-eliminated vertex, so there is only one of two options and the choice ratio is $\frac{1}{2}$. For the last vertex, the choice ratio is 1, and multiplied together, we get $\frac{1}{3}$. For a triangle, we can freely choose any vertex at first (1), but then both of the vertices that are left are adjacent to the eliminated vertex. Thus, the algorithm has to move to the next level of the elimination tree, and both vertices can be chosen again (1). When we consider the problems with more than three variables, the average choice ratio shifts towards lower values - in particular, there is now a large number of problems with a choice ratio close to 0. This is also to be expected, as the denominator of the degree of reordering grows very quickly with the number of variables. However, even with this large vertex count, we can observe a relatively large number of problems with a degree of reordering of one. This can be explained by the fact that the degree of reordering in E-Tree is always one for a clique: Since every vertex is simplical, any of them can be chosen for elimination. However, since it is also adjacent to all other vertices in the graph, the next vertex has to be on the next level, which means that the choice is not restricted to vertices not adjacent to the eliminated vertex. Since every polynomial adds a clique of its variable set into the associated graph, it is expected that fully connected graphs are rather common - one polynomial including all variables would be sufficient.

The resulting effect is that in many cases, the algorithm tends to behave exactly like Triangular. This can be confirmed when we filter the problems to contain only those where the choice ratio in E-Tree is one. There are 3579 total problems with that property. Of those, Triangular managed to solfe 3198, while ChordalTriangular could solve 3197. In comparison, the pure Chordal strategy could only solve 2986 problems from this subset.

When the graph is not triangular, the MCS-M algorithm is executed before the elimination tree is constructed. For this algorithm, the degree of reordering was evaluated as well, but it turned out to be rather low in general: For the problems with at least four vertices, we record a mean of $5.126 \cdot 10^{-4}$ and a median of $1.664 \cdot 10^{-49}$, which means that half of the problems have an even lower degree of reordering. It is unclear whether the choices in MCS-M tend to be more restricted in general, or the vertex count distribution for non-chordal graphs simply tends towards larger values than for chordal graphs. It is assumed that both aspects play a role here, but more research can be done. In particular, other completion algorithms could be explored, since a high degree of reordering, coupled with a good secondary ordering, might have a higher impact than the minimality of the chordal completion.

5.3.4 Analyzing the Projection Polynomial Sets

To try to gain some insight into the projection polynomial sets that are generated with our variable orderings, we collected various metrics of each projection level for each CAD call. In table B.3, we list the average size, maximum degree and sum of degrees of the projection polynomial sets for the first three projections. Overall, the effects of the incremental CAD are very noticeable here: We can see that level 0 is the “largest” by all three metrics, and going to level 1, the projection sets get significantly smaller in size and degree. For a regular CAD, we would expect the opposite - in general, the degree and projection size can grow quadratically. When we compare individual solvers, we can see some differences in behavior, but there are few patterns that can be explained by the variable orderings. Therefore, we will not perform further analysis on the properties of these polynomial sets.

Chapter 6

Conclusion

6.1 Summary

In this thesis, two previously presented variable orderings for CAD were introduced and compared: The Triangular ordering [EBDW14], which is used in the Maple RegularChains library, and a new ordering based on chordal graphs, first presented by Li et al. in [LXZZ21]. We have seen that the Triangular ordering chooses variables primarily based on their degree in the input polynomials, while the Chordal ordering uses an associated graph to analyze the connections of variables in the polynomial set. We presented two approaches to incorporate degree information into the Chordal ordering. For one approach, we used the Triangular ordering to make decisions at those points where the Chordal ordering would allow multiple choices for a variable. In a second approach, the associated graph structure was extended with edge labels, giving a bound on the degree of a variable (vertex) in some polynomials (that are represented by the edge). The new procedure greedily searches for a vertex in each step that would minimize the increase in the edge values and updates the edge labels when the vertex is eliminated. We have proven that the elimination procedure keeps the edge labels accurate, in the sense that their values continue to be an upper bound on the degree in the new polynomials that could be generated. To evaluate the presented orderings in an SMT-solving context, SMT-RAT was used to build variants of a CAD-based SMT solver, which were then executed on the QF_NRA problem set from SMT-LIB. In the evaluation we have seen that the combination of the Chordal and Triangular orderings performs better than a pure implementation of either, which suggests that the new ordering is successful in combining the different information that is captured by the two orderings. The new GDCC ordering also manages to offer better overall solving performance than the Chordal ordering in our experiments, although it is still worse than the direct Chordal-Triangular combination. However, a deeper analysis proved to be difficult, due to the incremental nature of the solver and the incompleteness of Proj_{mc}.

6.2 Future Work

While the experiments performed here were useful for gathering an overall impression on the performance of the orderings in an SMT context, the gathered data turned out to be insufficient for various reasons. Due to the incremental CAD implementation in SMT-RAT, it is difficult to see how the heuristics influence the performance of individual CAD calls. To address this, further experimentation should be done - in particular, it could be interesting to gather the CAD subproblems that are generated during SMT solving and evaluate the CAD on these problems individually. With this approach, one could guarantee that all heuristics are equally evaluated on all problems, so the resulting data might be more meaningful. In the presented experiments, the variable ordering was constructed once on the first call on the input constraints for that call. However, due to the incremental nature, the following calls will introduce additional constraints, which were not considered when the ordering was built. More research could be done on the performance of the orderings for these subsequent calls, to see whether a given ordering stays close to optimal when more constraints are added. Should the performance under a given ordering become suboptimal at some point, the CAD could be reset to allow for reordering of the variables. The condition for the reset is another heuristic choice, on which research can be done. Alternatively, the full constraint set could be considered in the ordering, with the hopes of finding a good ordering for all CAD calls. Another aspect of incrementality implemented into the `NewCADModule` is realized by the fine-grained projection and lifting phase, which could be modified to implement new heuristics for polynomial choice during the projection. It might be possible to couple a variable ordering heuristic with a polynomial choice heuristic for incremental CAD - for example, the chordal ordering could be paired with a projection heuristic that postpones projections which would add fill edges in the associated graph if it is non-chordal.

The incompleteness of McCallum's projection operator was another issue in the evaluation, since the large number of `UNKNOWN` results leads to various inaccuracies when comparing data. To address this issue directly, a similar experiment could be performed using a complete projection operator. In the resulting dataset, a strong correlation between chordality of the associated graph and the ratio of `UNKNOWN` results to correct `SAT`/ `UNSAT` answers could be observed. This correlation could be researched in more detail: If a causal link can be established between chordality and the likelihood of an incomplete CAD using Proj_{mc}, a new heuristic based on this effect could allow the solver to postpone a CAD computation if it is unlikely to result in a definitive answer.

Furthermore, the new orderings could be explored in different contexts: For SAT solving in QF_NRA, new techniques like Cylindrical Algebraic Coverings or the use of CAD in an MCSAT framework are active topics of research that might also benefit from better variable orderings. Also, when CAD is applied for quantifier elimination, a full CAD has to be computed in general. In this case, the effect of the variable ordering might be greater than in the SMT solving context, so it could be particularly interesting to evaluate the new operators in this context.

In this thesis, a new labelling for the associated graph was proposed with the GDCC algorithm. There are many possibilities for extending the idea of this algorithm: The labelling could be changed, for example to include information about the size of the polynomials associated with an edge. Correspondingly, different heuristics for choosing a vertex might be explored.

Bibliography

[BBHP04] Berry, Anne ; Blair, Jean R. S. ; Heggernes, Pinar ; Peyton, Barry W.: Maximum Cardinality Search for Computing Minimal Triangulations of Graphs. In: *Algorithmica* 39 (2004), August, Nr. 4, 287–298. `http://dx.doi.org/10.1007/s00453-004-1084-3`. – DOI 10.1007/s00453–004–1084–3. – ISSN 0178–4617, 1432–0541

[BD07] Brown, Christopher W. ; Davenport, James H.: The Complexity of Quantifier Elimination and Cylindrical Algebraic Decomposition. In: *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation - ISSAC '07*. Waterloo, Ontario, Canada : ACM Press, 2007. – ISBN 978–1–59593–743–8, 54

[BDE$^+$16] Bradford, Russell ; Davenport, James H. ; England, Matthew ; McCallum, Scott ; Wilson, David: Truth Table Invariant Cylindrical Algebraic Decomposition. In: *Journal of Symbolic Computation* 76 (2016), September, 1–35. `http://dx.doi.org/10.1016/j.jsc.2015.11.002`. – DOI 10.1016/j.jsc.2015.11.002. – ISSN 07477171

[BR04] Brown, Christopher W. ; Road, C H.: Cylindrical Algebraic Decomposition - Tutorial Notes. (2004), Juli, S. 14

[Bro01] Brown, Christopher W.: Improved Projection for Cylindrical Algebraic Decomposition. In: *Journal of Symbolic Computation* 32 (2001), November, Nr. 5, 447–465. `http://dx.doi.org/10.1006/jsco.2001.0463`. – DOI 10.1006/jsco.2001.0463. – ISSN 07477171

[CMXY09] Chen, Changbo ; Maza, Marc M. ; Xia, Bican ; Yang, Lu: *Computing Cylindrical Algebraic Decomposition via Triangular Decomposition.* `http://arxiv.org/abs/0903.5221`. Version: März 2009

[Col75] Collins, George E.: Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decompostion. Version: 1975. `http://dx.doi.org/10.1007/3-540-07407-4_17`. In: Goos, G. (Hrsg.) ; Hartmanis, J. (Hrsg.) ; Brinch Hansen, P. (Hrsg.) ; Gries, D. (Hrsg.) ; Moler, C. (Hrsg.) ; Seegmüller, G. (Hrsg.) ; Wirth, N. (Hrsg.) ; Brakhage, H. (Hrsg.): *Automata Theory and Formal Languages 2nd GI Conference Kaiserslautern, May 20–23, 1975* Bd. 33. Berlin, Heidelberg : Springer Berlin Heidelberg, 1975. – DOI 10.1007/3–540–07407–$4_1 7. - -ISBN 978-$ $-3- -540- -07407- -6978- -3- -540- -37923- -2, 134- -183$

[EBDW14] England, Matthew ; Bradford, Russell ; Davenport, James H. ; Wilson, David: Choosing a Variable Ordering for Truth-Table Invariant Cylindrical Algebraic Decomposition by Incremental Triangular Decomposition. In: Hong, Hoon (Hrsg.) ; Yap, Chee (Hrsg.): *Mathematical Software – ICMS 2014*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2014. – ISBN 978–3–662–44199–2, S. 450–457

[FG65] Fulkerson, Delbert ; Gross, Oliver: Incidence Matrices and Interval Graphs. In: *Pacific Journal of Mathematics* 15 (1965), September, Nr. 3, 835–855. http://dx.doi.org/10.2140/pjm.1965.15.835. – DOI 10.2140/pjm.1965.15.835. – ISSN 0030–8730, 0030–8730

[JK82] Jess ; Kees: A Data Structure for Parallel L/U Decomposition. In: *IEEE Transactions on Computers* C-31 (1982), März, Nr. 3, 231–239. http://dx.doi.org/10.1109/TC.1982.1675979. – DOI 10.1109/TC.1982.1675979. – ISSN 0018–9340

[KA20] Kremer, Gereon ; Abraham, Erika: Fully Incremental Cylindrical Algebraic Decomposition. In: *Journal of Symbolic Computation* 100 (2020), September, 11–37. http://dx.doi.org/10.1016/j.jsc.2019.07.018. – DOI 10.1016/j.jsc.2019.07.018. – ISSN 07477171

[(kr23] (krisi0903), Kristian C.: *SMT-RAT - Satisfiability-Modulo-Theories Real Algebra Toolbox*. https://github.com/krisi0903/smtrat/. Version: Januar 2023

[Kre20] Kremer, Gereon: *Cylindrical Algebraic Decomposition for Nonlinear Arithmetic Problems*. Aachen, RWTH Aachen University, Diss., 2020. http://dx.doi.org/10.18154/RWTH-2020-05913. – DOI 10.18154/RWTH–2020–05913. – 1 Online–Ressource (204 Seiten) : Illustrationen, Diagramme S.

[LXZZ21] Li, Haokun ; Xia, Bican ; Zhang, Huiying ; Zheng, Tao: *Choosing the Variable Ordering for Cylindrical Algebraic Decomposition via Exploiting Chordal Structure*. http://arxiv.org/abs/2102.00823. Version: Februar 2021

[McC98] McCallum, Scott: An Improved Projection Operation for Cylindrical Algebraic Decomposition. In: Caviness, Bob F. (Hrsg.) ; Johnson, Jeremy R. (Hrsg.): *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Vienna : Springer Vienna, 1998. – ISBN 978–3–7091–9459–1, S. 242–268

[Par61] Parter, S.: The Use of Linear Graphs in Gauss Elimination. In: *SIAM Review* 3 (1961), April, Nr. 2, 119–130. http://dx.doi.org/10.1137/1003021. – DOI 10.1137/1003021. – ISSN 0036–1445, 1095–7200

[Pot88] Pothen, A.: *The Complexity of Optimal Elimination Trees*. Pennsylvania State University, Department of Computer Science, 1988 (Technical Report). https://books.google.de/books?id=PfyjtgAACAAJ

[smt] *SMT-LIB The Satisfiability Modulo Theories Library*. https://smtlib.cs.uiowa.edu/benchmarks.shtml

[smt22] *SMT-RAT - Satisfiability-Modulo-Theories Real Algebra Toolbox*. Theory of Hybrid Systems group @ RWTH Aachen University. https://github.com/ths-rwth/smtrat. Version: November 2022

[TY84] Tarjan, Robert E. ; Yannakakis, Mihalis: Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs. In: *SIAM Journal on Computing* 13 (1984), Nr. 3, 566–579. `http://dx.doi.org/10.1137/0213035`. – DOI 10.1137/0213035

[VKÁ07] Viehmann, Tarik ; Kremer, Gereon ; Ábrahám, Erika: Comparing Different Projection Operators in the Cylindrical Algebraic Decomposition for SMT Solving. In: *[2nd International Workshop on Satisfiability Checking and Symbolic Computation, SC2, 2017-07-29 - 2017-07-29, Kaiserslautern, Germany / Edited by Matthew England, Vijay Ganesh]* Bd. 1974. Aachen, Germany : RWTH Aachen, 2017-07-29, 2017-07 (CEUR Workshop Proceedings), 15 Seiten

Appendix A

Source Code Listings

Listing A.1: Reference CAD Settings

```
// filename: src/smtrat-modules/NewCADModule/NewCADSettings.h.in
// Settings to test different variable orderings
//

// ... some code omitted ...

struct NewCADBaseVariableOrderingSettings : NewCADBaseSettings, cad::
    IncrementalityFU, cad::SampleCompareLT, cad::ProjectionOrderDefault,
    cad::ProjectionMcCallum {};

struct NewCADSettingsChordal: NewCADBaseVariableOrderingSettings {
  static constexpr auto moduleName = "NewCADModule<NewCADChordal>";
  static constexpr cad::variable_ordering::VariableOrdering
      variableOrdering = &smtrat::cad::variable_ordering::
      chordal_vargraph_elimination_ordering<smtrat::cad::variable_ordering
      ::ChordalOrderingSettingsBase>;
};

// ... some code omitted ...
```

Listing A.2: Reference SMT Strategy

```
// filename: src/smtrat-strategies/strategy/CADVOTest.h.template
#pragma once

#include <smtrat-solver/Manager.h>

#include <smtrat-modules/NewCADModule/NewCADModule.h>
#include <smtrat-modules/SATModule/SATModule.h>

namespace smtrat
{
  class CADChordal: public Manager
  {
    public:
    CADVOTest(): Manager() {
      setStrategy({
        addBackend<SATModule<SATSettings1>>({
          addBackend<NewCADModule<__CADSETTINGS__>>()
        })
        });
      }
    };
  } // namespace smtrat
```

Appendix B

Experimental Data

Algorithm 6 Intersection

```
procedure INTERSECTION(P)
    $R \leftarrow \{\textsc{Solver}(P) \mid \textsc{Solver} \in S\}$
    if MEMOUT $\in R$ then
        return MEMOUT
    end if
    if TIMEOUT $\in R$ then
        return TIMEOUT
    end if
    if UNKNOWN $\in R$ then
        return UNKNOWN
    end if
    if UNSAT $\in R$ then
        return UNSAT
    end if
    if SAT $\in R$ then
        return SAT
    end if
end procedure
```

Algorithm 7 Union

```
procedure UNION(P)
    $R \leftarrow \{\textsc{Solver}(P) \mid \textsc{Solver} \in S\}$
    if SAT $\in R$ then
        return SAT
    end if
    if UNSAT $\in R$ then
        return UNSAT
    end if
    if UNKNOWN $\in R$ then
        return UNKNOWN
    end if
    if TIMEOUT $\in R$ then
        return TIMEOUT
    end if
    if MEMOUT $\in R$ then
        return MEMOUT
    end if
end procedure
```

In the algorithms above, P is a set of problems, and $S = \{$CHORDAL, CHORDALTRIANGULAR PSEUDORANDOM, CHORDALNOETREE, DMFILL, DMFILLCHORDAL,TRIANGULAR$\}$ is the set of solvers

solver	SAT	UNSAT	UNKNOWN	TIMEOUT	MEMOUT	SOLVED	ANSWERED
Overall results							
INTERSECTION	3949	3377	647	3402	759	7326	7973
CHORDAL	4166	3517	506	3501	444	7683	8189
PSEUDORANDOM	4161	3538	445	3600	390	7699	8144
CHORDALNOETREE	4216	3631	400	3449	438	7847	8247
CHORDALTRIANGULAR	4274	3614	454	3358	434	7888	8342
DMFILL	4271	3631	386	3394	452	7902	8288
DMFILLCHORDAL	4271	3631	390	3508	334	7902	8292
TRIANGULAR	4329	3648	358	3322	477	7977	8335
UNION	4450	3780	228	3486	190	8230	8458
$\lvert\texttt{vars}\rvert \leq \mathbf{3}$							
INTERSECTION	3293	1893	354	589	0	5186	5540
CHORDAL	3405	1971	295	458	0	5376	5671
PSEUDORANDOM	3445	2019	201	464	0	5464	5665
CHORDALTRIANGULAR	3490	2033	301	305	0	5523	5824
CHORDALNOETREE	3451	2082	192	404	0	5533	5725
DMFILL	3507	2081	177	364	0	5588	5765
DMFILLCHORDAL	3509	2081	177	362	0	5590	5767
TRIANGULAR	3551	2085	197	296	0	5636	5833
UNION	3634	2185	84	226	0	5819	5903
$\lvert\texttt{vars}\rvert > \mathbf{3}$							
INTERSECTION	656	1484	293	2813	759	2140	2433
PSEUDORANDOM	716	1519	244	3136	390	2235	2479
CHORDAL	761	1546	211	3043	444	2307	2518
DMFILLCHORDAL	762	1550	213	3146	334	2312	2525
CHORDALNOETREE	765	1549	208	3045	438	2314	2522
DMFILL	764	1550	209	3030	452	2314	2523
TRIANGULAR	778	1563	161	3026	477	2341	2502
CHORDALTRIANGULAR	784	1581	153	3053	434	2365	2518
UNION	816	1595	144	3260	190	2411	2555

Table B.1: Distribution of solver results on the full QF_NRA problem set by solver. The table is split into three sub-tables: The first shows the distribution for the whole problem set, while the lower two show the results for the subset with less than four and at least four real arithmetic variables respectively

cadrun	time	combined deg.	max. deg.	sum of deg.	size
0	79.674	5.696	1.065	3.391	10.457
1	38.196	8.761	1.761	5.478	11.935
2	5.174	10.804	1.826	7.065	13.196
3	15.913	12.087	1.804	8.261	14.283
4	24.587	13.391	1.826	9.283	15.152
5	13.913	14.109	1.804	9.913	15.783
6	12.457	14.391	1.783	10.087	16.261
7	13.804	14.478	1.804	10.348	16.739
8	83.022	14.065	1.848	10.087	17.130
9	27.957	13.826	1.804	9.500	17.239

Table B.2: Average values for properties of the polynomial sets that were seen in the i-th CAD call with the PSEUDORANDOM ordering. In this table, all problems (46) in the benchmark set that lead to at least 10 CAD calls before termination are taken into account.

level	CHOR.	CHOR.NOETREE	CHOR.TRI.	DMFILL	DMFILLCHOR.	PRAND.	TRI.
Projection size							
0	7.469	7.469	7.469	7.707	7.707	7.707	7.707
1	1.301	1.264	1.271	1.254	1.254	1.352	1.275
2	1.246	1.236	1.250	1.251	1.246	1.396	1.257
3	1.329	1.316	1.317	1.319	1.319	1.430	1.316
Maximum degree							
0	2.004	1.946	1.871	1.896	1.896	2.106	1.658
1	1.308	1.274	1.277	1.280	1.280	1.360	1.094
2	1.205	1.254	1.146	1.219	1.217	1.355	1.138
3	1.193	1.120	1.085	1.183	1.183	1.240	1.119
Sum of degrees							
0	3.378	3.254	3.275	3.055	3.055	3.742	3.475
1	2.026	1.927	2.013	1.943	1.943	2.300	2.245
2	2.018	2.333	1.908	2.180	2.179	2.635	2.662
3	2.183	2.356	1.967	2.461	2.461	2.473	2.460

Table B.3: Average values of properties of the polynomial sets. Level i refers to the set after i projections, so level 0 refers to the input polynomial set. The average is computed across all problems that were solved by all solvers with at least four variables, to ensure that every table cell is computed with values from the same set of problems. To keep the table small enough to fit on the page, some strategies were abbreviated: TRI. is short for TRIANGULAR, CHOR. is short for CHORDAL and P.RAND. is short for PSEUDORANDOM

solver	SAT	UNSAT	UNKNOWN	SOLVED	ANSWERED	total
All						
Chordal	4500	2039	2161	6539	8700	10152
Pseudorandom	4532	2077	2093	6609	8702	10099
ChordalTriangular	4575	2133	1960	6708	8668	10144
DMFillChordal	4574	2152	2196	6726	8922	10151
DMFill	4591	2153	1916	6744	8660	10039
ChordalNoETree	4623	2157	2025	6780	8805	10239
Triangular	4831	2175	1373	7006	8379	10117
#vars $\leq$ 3						
Chordal	3447	1446	418	4893	5311	5580
ChordalTriangular	3505	1506	413	5011	5424	5581
Pseudorandom	3519	1505	289	5024	5313	5580
ChordalNoETree	3530	1568	288	5098	5386	5586
DMFillChordal	3558	1559	261	5117	5378	5577
DMFill	3558	1561	261	5119	5380	5578
Triangular	3600	1563	271	5163	5434	5582
#vars $>$ 3						
Pseudorandom	1013	572	1804	1585	3389	4519
DMFillChordal	1016	593	1935	1609	3544	4574
DMFill	1033	592	1655	1625	3280	4461
Chordal	1053	593	1743	1646	3389	4572
ChordalNoETree	1093	589	1737	1682	3419	4653
ChordalTriangular	1070	627	1547	1697	3244	4563
Triangular	1231	612	1102	1843	2945	4535

Table B.4: Distribution of results returned by the first CAD call across all problems from the benchmark set, split by the number of variables

solver	SAT	UNSAT	UNKNOWN	SOLVED	ANSWERED	total
Chordal						
Pseudorandom	644	399	187	1043	1230	1393
ChordalNoETree	644	406	188	1050	1238	1408
Chordal	643	408	186	1051	1237	1396
DMFill	649	407	177	1056	1233	1395
DMFillChordal	647	410	177	1057	1234	1394
Triangular	667	424	134	1091	1225	1393
ChordalTriangular	676	439	121	1115	1236	1394
Nonchordal						
DMFillChordal	12	27	1693	39	1732	1865
ChordalTriangular	19	30	1383	49	1432	1861
DMFill	27	29	1416	56	1472	1750
Chordal	35	27	1516	62	1578	1844
Pseudorandom	47	25	1539	72	1611	1828
ChordalNoETree	74	25	1506	99	1605	1903
Triangular	189	30	925	219	1144	1829

Table B.5: Distribution of results returned by the first CAD call across all problems from the benchmark set with at least three variables, split by chordality.