Alcom-FT Technical Report Series
ALCOMFT-TR-03-33

# On consistency and width notions for constraint programs with algebraic constraints

K. Meer*

Department of Mathematics and Computer Science
Syddansk Universitet, Campusvej 55, 5230 Odense M, Denmark
e–mail: meer@imada.sdu.dk

**Abstract.** Width notions have been studied for constraint satisfaction problems in order to exhibit subclasses of problems which allow backtrack-free solution algorithms. Among the most prominent such notions is the tree-width of the constraint graph studied, for example, by Freuder [5, 6]. However, Freuder's results heavily rely on constraint programs over finite domains, where each constraint is given as a list of admissible tuples and therefore fails, for example, if continuous domains are considered. Faltings [4] introduced an arc consistency notion for constraints over continuous domains that are given in a more complicated form using formulas $c(x, y) \geq 0$ for continuously differentiable functions $c$. He then showed for such binary constraints how arc consistency can be established and guarantees solvability of tree-structured problems.
In this paper we want to study a generalization of Freuder's and Faltings' notions to problems with algebraic constraints. We show that an analog notion of $k$-consistency guarantees backtrack-free solution algorithms for tree-structured problems, but argue that already for binary constraints and a tree as structure of the constraint graph there arise unavoidable complexity problems in achieving $k$-consistency. We then propose a new width notion based on [9] which in certain situations even allows to include global constraints without yielding a complexity explosion - something not true within the above mentioned setting.

**Key-words:** algebraic constraint satisfaction problems, width, consistency, backtrack-free algorithms

## 1 Introduction

An important branch of the analysis of constraint programs is the study of consistency notions. Certain local and global consistency notions were introduced. Here, one goal is to find subclasses of constraint programs for which validity of

* partially supported by EU ESPRIT project NeuroCOLT2 and by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

(some of) the former imply validity of (some of) the latter. Such results, for example, can be used in order to either perform a backtrack-free solution algorithm for the constraint program or to at least simplify the problem.

Among local consistency notions we can find node consistency, arc consistency, path consistency and, most general, $k$-consistency, see [10, 5]. Freuder [5, 6] has studied $k$-consistency and related it to the structure of a so called (primal) constraint graph. The latter has the problem variables as vertices; two variables are joined by an edge iff they occur together in a single constraint. Introducing a width notion for the constraint graph Freuder was able to establish certain subclasses of constraint programs which can be solved without backtracking, see section 2. Freuder's approach heavily relies on the variable domains and the way constraints are given: for each variable $x_i$ there is a *finite* set $D_i$ as its domain; a constraint $c$ over variables $x_{i_1}, \ldots, x_{i_k}, i_j \in \{1, \ldots, n\}$ is then given by explicitly listing the elements of a corresponding subset of $D_{i_1} \times \ldots \times D_{i_k}$. This approach is problematic, for example, if one deals with continuous constraints or if it is not ad hoc possible to list a constraint because it is given in an algebraic form (say as solution of an equation). In [4], Faltings defines arc consistency for certain classes of continuous constraint satisfaction programs. The (binary) constraints there are given as formulas $c(x, y) \geq 0$, where $c(x, y) = 0$ determines a differentiable curve. For binary problems whose constraint graph is a tree Faltings presents a finite algorithm yielding arc consistency (which in this situation implies global consistency). However, non-binary programs are not treated.

Part of the present paper is devoted to generalize and homogenize both Freuder's and Faltings' approaches to constraint programs (both discrete and continuous) for which the constraints are given in an algebraic manner. By the latter, we mean formulas of the form $c(x_{i_1}, \ldots, x_{i_k}) \Delta 0$, where $c$ is a polynomial with real coefficients and $\Delta \in \{=, \neq, \geq \leq, >, <\}$. After recalling the main concepts of [6, 4] in section 2, we shall extend the notions of width of the constraint graph and $k$-consistency to such constraint programs and study their relation. We shall see that both above approaches can be captured. Even though in [4] there was given a refinement procedure to achieve arc consistency, it turns out that the related algorithm is problematic already in the binary case and for algebraic constraints (i.e. instead of the more general constraints Faltings is considering); in general, an exploding complexity cannot be avoided. This is due to problems resulting from quantifier elimination.
Another problem with the width notion used so far is the fact that it is only useful (w.r.t. efficient solution algorithms) if no global constraints are present in the program. We therefore consider in section 3 a new width notion for such constraint programs. The latter was introduced in [9] in an algebraic-complexity framework. We propose its study also in the area of constraint programming problems. In some situations, it allows backtrack-free algorithms even though global (algebraic) constraints occur.

## 2 Algebraic constraint satisfaction problems; width-notion and consistency

We start by recalling some basic definitions, including Freuder's width notion for finite domain programs and Faltings' arc consistency notion for certain continuous variable programs. We then extend it to general algebraic constraint programs.

### 2.1 Freuder's and Faltings' settings

We start with

**Definition 1** *a) A* constraint satisfaction problem *(CSP) $\mathcal{P}$ is given as a finite set $X := \{x_1, \ldots, x_n\}$ of variables which can take values from finite domains $D_1, \ldots, D_n$ together with a set $\mathcal{C} := \{c_1, \ldots, c_m\}$ of constraints. Each $c_i$ is given as a pair $(var(c_i), r_i)$, where $var(c_i) \subseteq X$ denotes the variables of constraint $c_i$ and $r_i \subseteq \bigotimes_{x_j \in var(c_i)} D_j$ is a relation representing the admissible tuples for constraint $c_i$. The points satisfying relation $r_i$ are listed one by one.*
*b) A* solution *of a CSP is an assignment $x_i := a_i \in D_i$ such that for each constraint $c_j$ the projection of $(a_1, \ldots, a_n)$ to the corresponding components belongs to $r_j$.* Solving a CSP *means to decide existence of a solution. (One could as well ask for finding at least one or all solutions.)*
*c) The* constraint graph *(also called* primal constraint graph*) of a CSP $\mathcal{P}$ is the graph $G = (V, E)$ with $V := X$ and two vertices $x_i, x_j$ are joined in $E$ iff there exists a constraint $c$ such that $\{x_i, x_j\} \subseteq var(c)$.*

In order to identify subclasses of CSPs being solvable without backtracking Freuder [5] studied the tree-width of the primal graph.

**Definition 2** *a) A* tree decomposition *of a graph $G = (V, E)$ is a tree $T = (V_T, E_T)$ satisfying the following conditions:*

*i) to each node $t \in V_T$ there corresponds a subset $V_t \subseteq V$;*
*ii) $\bigcup_{t \in V_T} V_t = V$;*
*iii) $\forall (i, j) \in E$ there exists a node $t \in V_t$ such that $\{i, j\} \subseteq V_t$;*
*iv) for each $i \in V$ the subgraph of $T$ induced by those $t \in V_T$ such that $i \in V_t$ is connected.*

*b) The* width *of a tree decomposition $T$ is given as* $\max\{|V_t|, t \in V_T\} - 1$. *The* tree-width *$twd(G)$ of a graph $G$ is the minimal width of one of its tree decompositions.*

In recent years the notion of tree-width has played an important role in the design of efficient algorithms for certain subclasses of hard problems, see [3] and the literature cited in there. In relation with CSPs, Freuder has shown the following (see also [1] for the issue of constructing decompositions):

**Theorem 3.** *(Freuder [6]) Let $k \in \mathbb{N}$ be a fixed constant. Given a CSP whose constraint graph has tree-width at most $k$, there exists linear (in the number of variables) time algorithms finding all solutions of the problem.*

The solution algorithm is based on climbing up an optimal decomposition tree, each step taking care of the tuples satisfying those constraints which only relate variables covered by $V_t$ for the current node $t$. This update heavily relies on the fact that the admissible tuples are listed explicitly (and thus, the domains have to be finite). Note as well that a single global constraint (i.e. a constraint including $O(n)$ many variables) results in a tree-width of order $n$ and thus forbids the application of the above theorem.

In order to obtain similar results as well for continuous CSPs Faltings [4] has introduced an arc consistency notion for the following type of problems:

**Definition 4** *A* continuous binary CSP *is given by a set $X := \{x_1, \ldots, x_n\}$ of variables with associated domains $D_1, \ldots, D_n, D_i \subseteq \mathbb{R}$ and a finite set $\mathcal{C}$ of constraints $c_i(x,y) \geq 0, 1 \leq i \leq m$, where each $c_i$ determines a differentiable curve $c_i(x,y) = 0$.*

*The* constraint graph *of such a problem is defined as above.*

Due to the fact that the functions $c_i$ are rather general, numerical issues enter into the analysis of such CSPs. The domains $D_i$ are represented by so called interval labels $L_{x_i}$ which describe a finite set of bounded intervals from which assignments for $x_i$ can be chosen. Arc consistency is then defined in the following way: the label $L_x$ of a variable $x$ is arc consistent with the label $L_y$ of variable $y$ iff $(\forall x_0 \in L_x)$ $(\exists\ y_0 \in L_y)$ such that for all constraints $c$ which relate $x$ and $y$ it is $c(x_0, y_0) \geq 0$. Faltings then shows that for constraint graphs of tree-width 1 (i.e. trees) arc consistency for all pairs of variables guarantees global consistency; moreover, arc consistency can be obtained by making use of a linear (in the number of variables) number of applications of a procedure REFINE(x,y). The latter procedure propagates admissible values for one of the variables taking the labels for the other one and considering all constraints in both.

### 2.2 Extension to algebraic CSPs; consistency and complexity

We shall now define and analyze algebraic CSPs relying on Faltings' approach. The special problems we consider are at the same time more restricted and more general ones. More restrictive because we only allow constraints involving polynomials instead of general differentiable functions; more general because we consider arbitrary arities in the constraints as well as a $k$-consistency notion.

The above mentioned refinement procedure from [4] is, when applied to a polynomial $c$, strongly related to a restricted kind of quantifier elimination. As we shall see, this actually is the reason why in certain situations the task to make a binary CSP arc consistent cannot be afforded efficiently even if the constraints are polynomials. Thus, reaching arc consistency with a linear number of refinement procedures in general does not result in suitable solution algorithms: the application of these operations might be as expensive as solving the entire CSP without taking into account any local consistency consideration, see Theorem 8 below.

Let us first define the corresponding notions.

**Definition 5** *a) An* algebraic constraint satisfaction problem $\mathcal{P}$ *is given as a tuple $\mathcal{P} = (X, \psi_1, \ldots, \psi_{|X|}, c_1, \ldots, c_m)$, where $X := \{x_1, \ldots, x_n\}$ is a finite set of variables each of which ranging over a domain $D_i \subseteq \mathbb{R}$. Every $D_i$ is (finitely) represented via a first-order formula $\psi_i(x)$ over the real numbers, i.e. a Boolean combination of (in-)equalities $q(x) \Delta 0$ for polynomials $q$ and $\Delta \in \{=, \neq, \geq, \leq, >, <\}$. Then $D_i = \{x \in \mathbb{R} \mid \psi_i(x)\}$. The finitely many constraints $c_i$ are of the form $c_i(x_{i_1}, \ldots, x_{i_{j_i}}) \Delta_{c_i} 0$, where $c_i$ is a real polynomial and $\Delta_{c_i}$ is as above. Again, $var(c_i) = \{x_{i_1}, \ldots, x_{i_{j_i}}\}$ denotes the variables constraint $c_i$ depends on.*
*b) The* size *of an algebraic CSP $\mathcal{P}$ is given as*

$$size(\mathcal{P}) := \sum_{i=1}^{m} size(c_i) + \sum_{i=1}^{n} size(\psi_i).$$

*Here, the size of a constraint $c_i$ in $j_i$ variables is defined to be $j_i \cdot degree(c_i)$, and the size of a first-order formula $\psi$ is the number of involved basic (in-)equalities multiplied with the maximal degree occurring.*

*Remark 1.* The size of a first-order formula could be defined in a different manner; for example, if all coefficients are rationals one could include the bit-sizes of these coefficients as well. We restrict ourselves to the above algebraic measure which is appropriate to explain the ideas.

*Example 1.* a) We want to express a finite CSP in our setting. If the finite domains are given by $D_1, \ldots, D_n$ each $D_i$ can be represented by a first-order formula

$$\psi_i(x_i) \;\equiv\; \prod_{a \in D_i} (x_i - a) = 0$$

of degree $|D_i|$. A relation $r$ in variables $x_1, \ldots, x_s$ can be described by a polynomial equation

$$0 = p_r(x_1, \ldots, x_s) := \prod_{(a_1, \ldots, a_s) \in r} \sum_{i=1}^{s} (x_i - a_i)^2$$

of degree $2 \cdot |r|$. Supposing $d$ to be an upper bound on the domains' cardinalities the size of the entire algebraic CSP with $m$ constraints is of order $O(m \cdot d^n)$. If all constraints relate at most $k$ different variables each, then the size reduces to $O(m \cdot d^k)$.

b) A system of linear inequalities can be represented in a straightforward manner by defining every inequality as a constraint. The domains can either be described via a tautology (if we are looking for real solutions) or as in part a) if we look for 0-1 or other integral solutions.

**Definition 6** *Let $k \in \mathbb{N}$ and an algebraic CSP $\mathcal{P} = (X, \psi_1, \ldots, \psi_n, c_1, \ldots, c_m)$ over $n$ variables be given. Problem $\mathcal{P}$ is* strongly $k$-consistent *iff for all constraints in at most $k$ many variables the following holds: w.l.o.g. let $x_1, \ldots, x_\ell$ denote an arbitrary set of $\ell$ variables, $\ell \leq k-1$. Then for all choices $x_i = a_i, 1 \leq i \leq \ell$ which are admissible (i.e. $\psi_i(a_i)$ is true) and which satisfy all constraints $c$ with $var(c) \subseteq \{x_1, \ldots, x_\ell\}$ and for any additional variable $y \notin \{x_1, \ldots, x_\ell\}$ there is an admissible assignment $y := a$ such that $(a_1, \ldots, a_\ell, a)$ satisfies all the constraints among those $\ell + 1$ variables.*

Though the above definition is straightforward, the necessity to represent domains in algebraic CSPs in a different manner than by elementwise listing results in severe differences and difficulties being not present in the finite case. We analyze some of the latter in the rest of this section.

The (primal) constraint graph of an algebraic CSP is defined as before.

**Theorem 7.** *Let $k \in \mathbb{N}$ be fixed.*

*a) Given an algebraic CSP $\mathcal{P}$ it is decidable in polynomial time in $size(\mathcal{P})$ whether $\mathcal{P}$ is strongly $k$-consistent.*

*b) If $\mathcal{P}$ is strongly $k$-consistent and if its constraint graph has tree-width at most $k-1$, then it is decidable in polynomial time whether $\mathcal{P}$ is solvable.*

*Proof.* Ad a) Let $\underline{x} := (x_{i_1}, \ldots, x_{i_\ell}), i_1 < \ldots < i_\ell$ denote a tuple of $\ell \leq k-1$ variables and let $C_{\underline{x}} := \{c_1, \ldots, c_s\}$ denote those constraints of $\mathcal{P}$ only depending on variables of $\underline{x}$. For sake of notational simplicity we write $c(\underline{x})\Delta_c 0$ for such a constraint $c \in \mathcal{C}_{\underline{x}}$, even though $c$ might not depend on all components of $\underline{x}$. Let $y$ be an arbitrary additional variable and denote by $C_{\underline{x} \cup \{y\}}$ the constraints among variables $\underline{x}$ and $y$. Consistency for $\underline{x}, y$ asks for truth of the following formula:

$$(*)\ \forall\ a_{i_1}, \ldots, a_{i_\ell}\ \left\{\rho_{\underline{x}}(a_{i_1}, \ldots, a_{i_\ell})\ \Rightarrow\ \exists\ a\ \rho_{\underline{x} \cup \{y\}}(a_{i_1}, \ldots, a_{i_\ell}, a)\right\}.$$

Here, for a subset $\tilde{X}$ of variables the formula $\rho_{\tilde{X}}$ denotes the conjunction of the domain formulas $\psi_i$ for $x_i \in \tilde{X}$ and the formulas given by any constraint $c$ that satisfies $var(c) \subseteq \tilde{X}$. Thus,

$$\rho_{\underline{x}}(a_{i_1}, \ldots, a_{i_\ell})\ \equiv\ \bigwedge_{j=1}^{\ell} \psi_{i_j}(a_{i_j})\ \wedge \bigwedge_{var(c) \subseteq \underline{x}} c(a_{i_1}, \ldots, a_{i_\ell})\Delta_c 0$$

and

$$\rho_{\underline{x} \cup \{y\}}(a_{i_1}, \ldots, a_{i_\ell}, a) \equiv \rho_{\underline{x}}(a_{i_1}, \ldots, a_{i_\ell})\ \wedge\ \psi_y(a)\ \wedge \bigwedge_{var(c) \subseteq \underline{x} \cup \{y\}} c(a_{i_1}, \ldots, a_{i_\ell}, a)\Delta_c 0.$$

Checking $(*)$ to hold can be done by eliminating the $\ell$ universal and the single existential quantifier. Note that $k$ is a constant and $\ell \leq k-1$, so we have to eliminate a constant number of quantifiers only. Applying well known results concerning quantifier elimination over the reals (see, for example, [12]) this can be accomplished in polynomial time w.r.t. $size(\mathcal{P})$ (and exponential time in $k$). Now a formula like $(*)$ has to be checked for all possible choices of $k$ many variables. Again, taking into account $k$'s role as a fixed constant, we have to perform the above elimination algorithm a polynomial number of times: there are $O(n^k)$ possible choices (actually, the latter only has to be done for those subsets of variables commonly occurring in a single constraint).

Ad b) Given $\mathcal{P}$ we consider, for each single variable $x_i$, the proposition

$$\theta_i \equiv \exists\ a\ \psi_i(a)\ \wedge \bigwedge_{var(c)=\{x_i\}} c\Delta_c 0$$

and eliminate in polynomial time the existential quantifier. Then $\mathcal{P}$ is solvable (under the assumption of strong $k$-consistency) iff

$$\bigwedge_{i=1}^{n} \theta_i$$

is a true proposition. The latter equivalence can be seen in exactly the same way as for discrete programs using a top-down evaluation in the decomposition tree. □

Whereas the above results seem to be similar to those for finite programs, the question how to obtain strong $k$-consistency for a given program results in difficulties.

**Theorem 8.** *Given as input an algebraic CSP $\mathcal{P}$ whose primal constraint graph has tree-width $\leq 2$, there is no general polynomial time (in $size(\mathcal{P})$) algorithm computing an equivalent arc consistent program. Here, we call a two algebraic CSPs equivalent if they have the same solution set.*

*Proof.* Consider the following algebraic CSP: there are $n+1$ many variables $x_0, \ldots, x_n$ with binary constraints $c_1, \ldots, c_n$, where

$$c_i(x_i, x_{i-1}) \ \equiv\ (x_i - x_{i-1}) \cdot (x_i - x_{i-1} - \frac{1}{10^i}) = 0\ \ ,\ 1 \leq i \leq n.$$

The domains are defined via

$$\psi_0(x_0)\ \equiv\ (x_0 - 1) \cdot (x_0 - 2) = 0$$

for $x_0$ and as $\mathbb{R}$ for the other variables (but could be a finite interval such as $[0, 10]$ as well). The size of $\mathcal{P}$ is of order $O(n)$.

An easy calculation now shows that there are $O(2^n)$ admissible values for variable $x_n$ which can be extended to solution vectors of $\mathcal{P}$. Thus, any first order-formula which describes the domain of $x_n$ in an equivalent, arc consistent program must have exponential size as well (a semi-algebraic set of $2^n$ discrete points cannot be described by a shorter formula). $\square$

The last result shows that even if the constraint graph of an algebraic CSP is well-structured efficiency problems can arise due to the computation of a description of the corresponding projection sets.

*Remark 2.* i) The above result does, of course, not mean that there is no efficient algorithm deciding solvability. It just addresses the question of designing such an algorithm along the line of establishing strong $k$-consistency.
ii) The importance of projection sets in the finite framwork was studied in [8]. Once again, the above problems do not occur in that setting, compare Example 1.

An additional condition sufficient to guarantee a better complexity performance for such problems can be derived as follows.

Suppose $T$ to be a tree decomposition of width $k$ for the constraint graph. W.l.o.g. assume $T$ to be binary, and if $t$ is a branch node in $T$ with children $t_1, t_2$, then the sets $V_{t_1}, V_{t_2}$ are equal. If $\tilde{t}$ is the only child of $t$ the sets $V_T$ and $V_{\tilde{t}}$ differ only in one element. All these conditions can be fulfilled without problems, see [3]. As, for example, described in [4] consistency can be obtained by first performing a bottom-up and thereafter a top-down computation in $T$. At each step we have to perform a restricted quantifier elimination. Suppose that we want to pass from a leaf $\tilde{t}$ in $T$ to its father $t$, and assume $V_t = \{x_1, \ldots, x_s, y\}, V_{\tilde{t}} = \{x_1, \ldots, x_s, z\}$. Let the first-order formulas $\tilde{c}(x_1, \ldots, x_s, z)$ and $c(x_1, \ldots, x_s, y)$ denote the conjunction of all constraints in $\mathcal{P}$ only involving variables $(x_1, \ldots, x_s, z)$ and $(x_1, \ldots, x_s, y)$, respectively. Given the defining formulas $\psi_v$ for the domain of each variable $v$, passing from $\tilde{c}$ to $c$ can be done by propagating a quantifier free description of the projection to $x_1, \ldots, x_s$ of the feasible set for

$$\tilde{c}(x_1, \ldots, x_s, z) \wedge \bigwedge_{i=1}^{s} \psi_{x_i} \wedge \psi_z$$

to the upper level, i.e. apply quantifier elimination to

$$\exists\ z\ \tilde{c}(x_1, \ldots, x_s, z) \wedge \bigwedge_{i=1}^{s} \psi_{x_i} \wedge \psi_z$$

and add the resulting formula, say $\rho_{\tilde{c}}(x_1, \ldots, x_s)$, by conjunction to $c$.

If we climb the tree, then other variables might be replaced. This results in an elimination in the last obtained formula of the variable to be replaced next. If a branch node is reached, by our initial assumption on $T$ we just have to join the corresponding sub-formulas by conjunction and potentially simplify the resulting one. After reaching the root a top-down computation is performed in the same manner, this time applying appropriate quantifier eliminations in

the reverse variable order (and to possibly different formulas). This process will finally give a $k$-consistent algebraic CSP.
The above algorithm runs in polynomial (linear in $n$) time if all first-order formulas computed in between have a size bounded by an (arbitrary) function in $k$. We have thus proven:

**Theorem 9.** *Let $k \in \mathbb{N}$ be fiexd. Under the above boundedness condition, an algebraic CSP can be transformed to an equivalent strongly $k$-consistent one in polynomial time. If, in addition, the constarint graph has tree-width $\leq k-1$ the algebraic CSP can be solved in polynomial time.*

Note that a direct application of the existing complexity estimates for quantifier elimination does not guarantee that during the above procedure formula sizes will remain polynomially bounded in the initial problem size (see Theorem 8).

We close this section by giving two examples where the above Theorem can be applied and thus, efficient algorithms exist on instances of bounded-width constraint graphs.

*Example 2.* a) The finite CSP setting (see Example 1) is captured by the above approach. Updating the describing formulas when moving bottom up or top down here results in removing certain tuples which turned out to be infeasible. With respect to the describing quantifier free formulas this leads to the deletion of certain factors in the polynomials. Therefore, the sizes of the formulas produced will never be longer than the initial formulas defining the variable domains, and are thus bounded by $O(m \cdot d^k)$. There is no blow up and the above arguments re-establish Freuder's result.
b) Consider a generalization of "simple temporal problems" studied in [2]. A constraint $c$ is given as a linear inequality of the form

$$a^T \cdot x + d \ \leq \ \tilde{a}^T \cdot x$$

for vectors $a, \tilde{a} \in \{0,1\}^n, d \in \mathbb{R}$. Quantifier elimination for a single variable can be done by Fourier-Motzkin elimination (see [13], and [11] for Fourier-Motzkin elimination in the framework of simple temporal problems). This method eliminates a variable $y$ in a set of formulas

$$a_i^T \cdot x + d_i \ \leq \ y \ \leq \ \tilde{a}_j^T \cdot x + \tilde{d}_j,$$

where $i \in I, j \in J, I, J$ are finite index sets, each $a_i, \tilde{a}_j \in \{0,1\}^n, d_i, \tilde{d}_j \in \mathbb{R}$, by replacing them with

$$a_i^T \cdot x + d_i \ \leq \ \tilde{a}_j^T \cdot x + d_j \ \ \forall\, i \in I, j \in J \ .$$

But the special form of the coefficients implies that for a set of $k$ many variables we shall never get more than $O(2^k)$ many inequalities. So for constraint graphs of bounded tree-width the algebraic CSP can be solved in linear time (even with computing an interval representation of all solutions).

Note that the above argument works in a complexity theoretic setting where only the number of arithmetic operations is counted. In such a framework it is an open problem whether (general) Linear Programming is polynomial time solvable.

A challenging problem is certainly provided by finding classes of algebraic CSPs for which the above condition on formula sizes is fulfilled.

## 3 A new width notion for algebraic CSPs

In the previous section we have seen that a straightforward generalization of the width notion for the constraint graph of algebraically given constraints leads to difficulties. In addition, this approach cannot be used as soon as global constraints (i.e. constraints combining $O(n)$ many variables) are present.
In this section we want to consider another width notion which was first introduced in [9] in an algebraic complexity framework. It takes into account the special algebraic form of the constraints. In addition, up to a certain extend also global constraints can be treated in a backtrack-free manner.

We present the main ideas behind this new notion. Instead of considering a constraint as an entity its special structure will be analyzed more precisely. As we are dealing with algebraic constraints, the latter refers to the structure of the monomials in a constraint such as $p(x_1,\dots,x_n)\Delta 0, p$ a polynomial. In particular, we are not so much interested in how variables are mixed in single constraints (as it is the case with Freuder's width notion) but how they are mixed within each monomial of a constraint. To a certain extend this point of view allows more general constraints to be treated efficiently, and even certain global ones can be admitted.

Let us now define the *algebraic width* of an algebraic CSP $\mathcal{P}$. Suppose the latter to be of the form $\mathcal{P} = (X, \psi_1, \dots, \psi_{|X|}, c_1, \dots, c_m)$, see Definition 5. We assume all constraints to have a degree at most $d \in \mathbb{N}$. We consider each constraint $c$ in $\mathcal{P}$ as a weighted hypergraph $(V, E), V := \{x_0, x_1, \dots, x_n\}, E_c \subseteq V^{d+1}$. The weights on each hyperedge $e$ of $E_c$ are given by a function $w_c : E \to \mathbb{R}$. This weight function represents the coefficients of monomials in $c$ in the following way: for a hyperedge $e := x_0^{\alpha_0} \cdot x_1^{\alpha_1} \cdot \ldots \cdot x_n^{\alpha_n}, \sum\limits_{i=0}^{n} \alpha_i = d$, the value $w_c(e)$ represents the coefficient of monomial $x_1^{\alpha_1} \cdot \ldots \cdot x_n^{\alpha_n}$ in $c$. Note that the vertex $x_0$ in $V$ is only used for dehomogenization of the given constraint.

**Definition 10** *Let $d \in \mathbb{N}$ be fixed and let $\mathcal{P} := (X, \psi_1, \dots, \psi_{|X|}, c_1, \dots, c_m)$ be given as above, all $c_i$ of degree at most $d$. Let $V := \{x_0, \dots, x_n\}, E_{c_i}$ be the set of hyperedges for constraint $c_i$ and $w_{c_i}$ its weight function, $1 \le i \le m$.*
*a) An* algebraic tree decomposition *of $\mathcal{P}$ is a tree $T = (V_T, E_T)$ satisfying the following conditions:*

*i) to each node $t \in V_T$ there corresponds a subset $V_t \subseteq \{c_1, \dots, c_m\} \cup V$;*
*ii) $\bigcup\limits_{t \in V_T} V_t = \{c_1, \dots, c_m\} \cup V$;*

*iii) for every constraintc and for all* $(x_{i_1}, \ldots, x_{i_d}) \in E_c$ *there exists a node* $t \in V_t$ *such that* $\{c, x_{i_1}, \ldots, x_{i_d}\} \subseteq V_t$;
*iv) for every* $i \in \{c_1, \ldots, c_m\} \cup V$ *the subgraph of* $T$ *induced by those* $t \in V_T$ *such that* $i \in V_t$ *is connected.*

*b) The* algebraic width *of a tree decomposition* $T$ *is given as* $\max\{|V_t|, t \in V_T\} - 1$. *The* algebraic tree-width $twd(\mathcal{P})$ *of* $\mathcal{P}$ *is the minimal width of one of its tree decompositions.*

*Remark 3.* A few remarks are necessary here.
i) The above way to define an algebraic width notion logically corresponds to the consideration of two-sorted structures: one sort being represented by the variables and the constraints, the other by the monomials involved in the constraints, see [9]. This is different from studying the primal graph of a constraint satisfaction program, which takes the variables as the only sort.
ii) Formalizing a CSP in different ways, especially with respect to the constraint graph, is not a new idea. A survey on such approaches in the framework of finite constraint programs and database theory can be found, for example, in [7]. There, a hypergraph width is studied. However, when applied to an algebraic setting, these notions create similar problems like those mentioned in section 2. For example, a single global constraint has a bounded hypertree width. This is, of course, in an algebraic setting not sufficient to guarantee efficient solution algorithms because already one global algebraic constraint can express an **NP**-hard problem. In addition, the hypertree-width still treats a whole constraint as an entity (which is, of course, appropriate if the constraint is listed elementwise). Therefore, for algebraic constraints we believe our notion to be more suitable taking into account also the structure of each single constraint.

The above width notion was introduced in [9] in order to deal with algebraic problems such as solving polynomial systems of equations over finite fields or the reals. In such a framework, it seems to be necessary to incorporate a significant part of logical methods (basically, an adaption of the theorem by Feferman and Vaught for the particular logical structures coming into play when describing such algebraic problems). In order to concentrate on the main ideas for using this approach to solve algebraic CSPs, here we will deal with the **NP**-complete 0-1 integer programming problem only.

**Theorem 11.** *Let* $k \in \mathbb{N}$ *be fixed. Let an algebraic CSP* $\mathcal{P} = (X, \psi_1, \ldots, \psi_{|X|}, c_1, \ldots, c_m)$ *be given such that*

*i) all the constraints* $c_i$ *are linear inequalities and all* $\psi_j$ *define the domain* $\{0,1\}$;
*ii)* $\mathcal{P}$ *has algebraic width at most* $k$;
*iii) the coefficients of the* $c_i$ *are integers in* $\{-q(|X|), \ldots, q(|X|)\}$ *for a polynomial* $q$.

*Then* $\mathcal{P}$ *can be solved without backtracking in linear time in the number* $n := |X|$ *of variables.*

*Proof.* The algorithm starts by producing a tree-decomposition $T$ of algebraic width $k$, this time w.r.t. the hypergraph representing the set of constraints. This can be done as usual, see [3]. The propagation will work bottom up along the nodes of $T$. However notice that, because of the fact that the width notion is based on monomials in a constraint, we have to take care about how a single constraint is decomposed as well by $T$. Consider a node in $T$ with the corresponding set $V_t, |V_t| \leq k+1$. Therefore, $V_t$ represents parts of constraints in a bounded number of variables together with monomials combining precisely these variables. Given a fixed bound $d$ on the degrees for the constraints, here $d=1$, the number of different monomials in those variables occuring in $V_t$ is constant w.r.t. the total number $n$ of variables. We now start the evaluation of these monomials and their sum on the substructure induced by $V_t$. It is important to note that on the substructures obtained by the decomposition the algebraic task to perform is that of partially evaluating the entire constraint (this might not be the same in general for arbitrary algebraic constraints, see the remarks following the proof).

Since we finally have to check an algebraic condition we have to store the different results intermediately obtained on the substructures for the different choices of assignments. The coefficient condition iii) on the $c_i$'s guarantees that at most polynomially many different subresults occur: if the possible assignments for the $x_i$ are taken from $\{0,1\}$, then there are only polynomially many subresults occuring when adding arbitrary monomials of a constraint. Therefore, if we climb up the decomposition tree and replace a variable already treated by a new one, then dealing with the new variable will not result in more than polynomially many different subresults. Backtracking as usual is avoided since a variable which was removed once will never reoccur. Finally, at the root we check whether there was an assignment meeting the required inequalities for all constraints. □

The above result is formulated in such a way that major problems with applying similar techniques to more general algebraic problems remain hidden. The approach can be extended to other algebraic CSPs, but then including a broader study of the logical framework in which such problems have to be expressible. The latter involves the introduction of a monadic second order logic on so-called meta-finite structures. This logic is used in order to specify which kind of algebraic problems can be treated with our approach. For the above 0-1 programming problem these difficulties lie quiet. For linear or quadratic programs where real solutions are looked for, the same becomes much more difficult due to the fact that the corresponding logical descriptions now include quantifiers over the reals ('there exists a real solution') instead of 'discrete' quantifiers (there exists a 0-1 solution). We postpone the treatment of such problems to an extended version. For the principle ideas behind such logical aspects related to our algebraic width notion see [9].

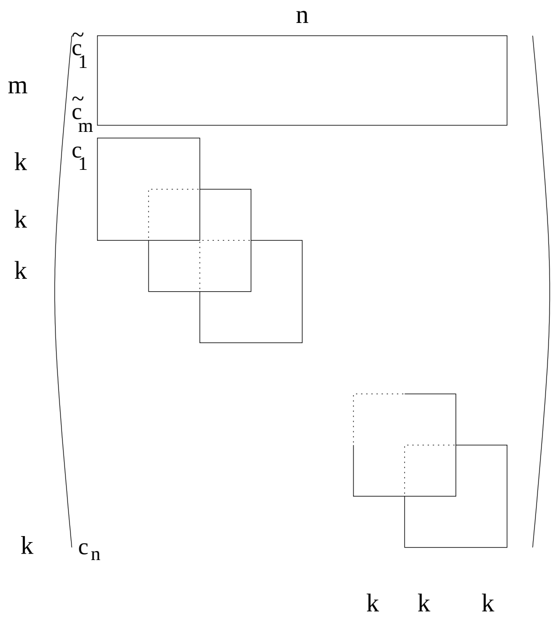
Let us mention that the above coefficient condition iii) is somehow related to the kind of problems we encountered in the proof of Theorem 8. Such a condition seems to be needed in order to avoid an explosion of the complexity even when variable backtracking does not occur.

We close with an example which shows that the above result captures certain constraint programs involving global constraints and thus extends the range of applicability of related techniques.

*Example 3.* Consider once more the generalization of simple temporal problems introduced in Example 2. The single constraints are of the form

$$(*)\ c(x_1,\ldots,x_n) := \sum_{i\in\tilde{S}} x_i - \sum_{j\in S} x_j + d \ \geq\ 0 \text{ for } S,\tilde{S}\subseteq\{1,\ldots,n\}, S\cap\tilde{S}=\emptyset$$

and satisfy condition iii) in Theorem 11.
We construct a class of such algebraic CSPs where the algorithm of Example 2 b) cannot be applied but Theorem 11 can. Let $m,k\in\mathbb{N}$ be constants independent of $n, k\cdot s = n$ for some $s\in\mathbb{N}, s>1$. First, consider a set of $m$ arbitrary (global) constraints $\tilde{c}_1,\ldots,\tilde{c}_m$ of form $(*)$. Next, consider quadratic blocks of $2k$ inequalities each, where block $i$ represents $2k$ constraints $c_{(i-1)k+1},\ldots,c_{(i+1)k}$ of form $(*)$ only involving variables $x_{(i-1)k+1},\ldots,x_{(i+1)k}$. The structure of this algebraic CSP is indicated in the next figure, where the boxes represent non-zero coefficients.




The usual tree-width of the primal constraint graph of such problems is $n$ because of the global constraints $\tilde{c}_1,\ldots,\tilde{c}_m$; however, it is easy to sea that the algebraic width is bounded by the constant $m+4k-1$ : a corresponding decomposition places $\tilde{c}_1,\ldots,\tilde{c}_m$; into each box $V_t$ of a decomposition for the algebraic program represented by the constraints $c_1,\ldots,c_n$.
If we look for a 0-1 solution or, more general, a solution with integer components in a set $\{-q(n),\ldots,q(n)\}$ for a polynomial $q$ we obtain an efficient solution algorithm.

In the present paper we studied generalization of the commonly used width notion for the (primal) constraint graph of a CSP to algebraic CSPs. Though, in principle, the definitions can easily be generalized, we showed that significant problems w.r.t. the complexity of propagation algorithms might occur even if the constraint graph is a tree. We then proposed the study of a new algebraic width notion in the framework of algebraic CSPs and showed how some problems can be solved efficiently by that approach, even though they might contain global constraints.

## References


1. H. Bodlaender, Treewidth: Algorithmic techniques and results. In I. Privara, P. Ruzicka, editors, Proc. 22nd Symposium *Mathematical Foundation of Computer Science*, volume 1295 of *Lecture Notes in Computer Science*, Springer, 19–36, 1997.
2. R. Dechter, I. Meiri, J. Pearl, Temporal constraint networks. Artificial intelligence 49 (1-3), 61–95 (1991).
3. R.G. Downey, M.F Fellows, Parametrized Complexity. Springer, 1999.
4. B. Faltings, Arc-consistency for continuous variables. Artificial Intelligence 65, 363–376, 1994.
5. E.C. Freuder, A sufficient condition for backtrack-free search. Journal of the ACM 29 (1), 61–95, 1982.
6. E.C. Freuder, Complexity of $k$-tree structured constraint satisfaction problems. Proc. of the 8th National Conference on AI, 4–9, 1990.
7. G. Gottlob, N. Leone, F. Scarcello, Hypertree Decompositions: A Survey. In J. Sgall, A. Pultr, P. Kolman, editors, Proc. 26nd Symposium *Mathematical Foundation of Computer Science*, volume 2136 of *Lecture Notes in Computer Science*, Springer, 37–57, 2001.
8. P. Jeavons, D. Cohen, M. Cooper, Constraints, Consistency, and Closure. Artificial Intelligence 101, 251–265, 1998.
9. J.A. Makowsky, K. Meer, Polynomials of bounded tree-width. To appear in: Proceedings of the Smalefest 2000, World Scientific.
10. K. Marriott, P.J. Stuckey, Programming with Constraints. The MIT Press, Cambridge, Massachusetts and London, England 1998
11. K. Meer, A simplified filtering algorithm for a global constraint involving a sum constraint and difference constraints. Preprint 2002.
12. J. Renegar, On the computational Complexity and Geometry of the first-order Theory of the Reals, I - III, *Journal of Symbolic Computation*, Vol. 13, 255–352, 1992.
13. A. Schrijver, Theory of Linear and Integer Programming. Wiley, Chichester, 1986.