Arc Consistency on n-ary Monotonic and Linear Constraints

Zhang Yuanlin and Roland H.C. Yap

School of Computing
National University of Singapore
3 Science Drive 2
Republic of Singapore 119260
{zhangyl,ryap}@comp.nus.edu.sg

Abstract. Many problems and applications can be naturally modelled and solved using constraints with more than two variables. Such n-ary constraints, in particular, arithmetic constraints are provided by many finite domain constraint programming systems. The best known worst case time complexity of existing algorithms (GAC-schema) for enforcing arc consistency on general CSPs is $O(ed^n)$ where d is the size of domain, e is the number of constraints and n is the maximum number of variables in a single constraint. We address the question of efficient consistency enforcing for n-ary constraints. An observation here is that even with a restriction of n-ary constraints to linear constraints, arc consistency enforcing is NP-complete. We identify a general class of monotonic n-ary constraints (which includes linear inequalities as a special case). Such monotonic constraints can be made arc consistent in time $\mathcal{O}(en^3d)$. The special case of linear inequalities can be made arc consistent in time $\mathcal{O}(en^2d)$ using bounds-consistency which exploits special properties of the projection function.

1 Introduction

Arc Consistency (AC) is an important technique for solving Constraint Satisfaction Problems (CSPs) [17]. A large part of the literature is thus on efficient algorithms for enforcing arc consistency on CSPs. The focus is usually on binary CSP where each constraint involves at most two variables. The well-known algorithms for arc consistency in binary CSPs include Waltz's filtering algorithm [30], AC-3 [17], AC-6 [4], AC-5 [28] and many others.

Constraint programming has shown that consistency techniques, in particular, AC-based methods are effective and useful for solving practical problems [27]. However, many real-life problems can be modelled naturally as a non-binary CSP where a constraint involves more than two variables. We call a constraint which involves an arbitrary number of variables an n-ary constraint. An n-ary CSP is then one where the maximum number of variables in constraints is at most n. Some typical examples of n-ary constraints include the *all different* constraint, the *cardinality* constraint [24] and linear arithmetic constraints. Such

n-ary constraints are provided by many constraint programming languages and libraries.

There are two main approaches to deal with n-ary CSPs. The first approach is to avoid altogether the question of an n-ary CSP. This is achievable since it is always possible to translate an n-ary CSP into a different binary CSP [10, 25]. The standard techniques in binary CSP can be used to solve the transformed CSP thus solving the original n-ary CSP also. A recent paper [1] is a detailed examination of the translation approach.

The second approach is to develop consistency techniques directly applicable to n-ary constraints. One direction is to extend techniques developed in the binary case for general n-ary CSPs. The other is to develop specialised techniques which can exploit the semantics of the particular n-ary constraints. Some representatives of first direction are as follows. Mackworth [18] generalized AC-3 to NC to deal with n-ary constraints. This is improved by GAC-4 [19] which is a generalization of AC-4. GAC-4 improves the complexity of NC, at the cost of a higher space complexity and a bad average time complexity. The time complexity of GAC-4 is $\mathcal{O}(ed^n)$ where e is the number of constraints and d is the size of the domain. We see that in contrast to their binary CSP AC versions, NC and GAC-4 may not be practical due to their high time complexity. A more efficient approach is the GAC-schema [5] based on *single support* and *multidirectionality* but it has the same worst case time complexity as GAC-4. The second direction is consistency algorithms for particular classes of constraints which can lead to more efficient algorithms, for example the global *all different* constraint and *cardinality* constraint [24].

The main contributions of this paper are the following. We address the problem of efficient consistency enforcing for n-ary constraints. An observation here is that even with a restriction of n-ary constraints to linear constraints, arc consistency enforcing becomes intractable. We identify a general class of monotonic n-ary constraints (which includes linear inequalities as a special case). Such monotonic constraints can be made arc consistent in time $\mathcal{O}(en^3d)$. The special case of linear inequalities can be made arc consistent using bounds-consistency which exploits special properties of the projection function in time $\mathcal{O}(en^2d)$.

This paper is organized as follows. First, we present some background material for n-ary CSP and the generalization of AC used here. We then formalize bounds based propagation as bounds-consistency for linear constraints. We give an efficient bounds-consistency algorithm for linear constraints. In Section 4, we look at arc consistency for linear inequalities and define a new class of monotonic constraints which is tractable. We then examine arc consistency for linear equations. Finally, we discuss related work.

2 Preliminaries

In this section we will give some definitions and notation for general n-ary CSPs [17, 19].

Definition 1. *An n-ary* Constraint Satisfaction Problem *(N, D, C) consists of a finite set of variables $N = \{1, \cdots, m\}$, a set of domains $D = \{D_1, \cdots, D_m\}$, where D_i is a finite set of values that i can take, and a finite set of constraints $C = \{c_X \mid X \subseteq N\}$, where each constraint c_X is a relation on variables of set X and thus c_X is a subset of $D_{i_1} \times D_{i_2} \times \cdots \times D_{i_l}$ where $i_k \in X, k \in \{1, \ldots, l\}$. The arity of the CSP is defined as $n = Max\{|X| \mid c_X \in C\}$.*

Throughout this paper, the number of variables is denoted by m, the maximum arity of constraints in the n-ary CSP is n, the size of largest domain is d, and the number of constraints is e. Thus, a binary CSP is simply a 2-ary CSP.

A constraint in an n-ary CSP may be defined and represented in a number of ways. It can be represented explicitly as a set of tuples (either allowed or disallowed), a conjunctive constraint, implicitly as an arithmetic expression, or by any predicate whose semantics is defined by a particular definition/program code. In this paper, we will use the notation c_X to represent both the form of a constraint and the set of tuples that satisfy the constraint.

Definition 2. *Given a CSP (N, D, C) and a constraint $c_X \in C$. We define a* solution *of constraint c_X to be any tuple $\langle v_{i_1}, \cdots, v_{i_n} \rangle \in c_X$. If c_X is empty, we say that there is no solution for c_X.*

We are now in a position to define arc consistency for n-ary CSPs. The following definition from Mackworth [17] is one natural generalization of arc consistency.

Definition 3. *Given an n-ary CSP (N, D, C), a constraint $c_X \in C$ is* arc consistent *with respect to D iff $\forall i \in X$ and $\forall v \in D_i$, v is a component of a solution of c_X in which case v is said to be* valid *with respect to c_X. A CSP (N, D, C) is arc consistent iff all $c_X \in C$ are arc consistent.*

In this paper, we will employ this particular definition of arc consistency for n-ary CSPS which is sometimes also called *hyper-arc consistency*. We remark that our definition of arc consistency is similar to relational arc consistency [26]. Enforcing higher consistency such relational path consistency on the n-ary CSPs is NP-complete in general (see Section 5).

The task of an arc consistency algorithm is then to remove those invalid values from the n variables in each constraint. In a binary CSP, the representation of a constraint may not be so important for this process. In the n-ary CSP case, the precise representation may fundamentally affect the efficiency of the arc consistency algorithm. For example, the *all different* constraint can be represented in a number of ways. Suppose that we represent the *all different* constraint using an explicit tuple representation as in GAC-4, the set of allowed tuples could be huge which may be impractical in terms of space and time. The GAC-schema of [5] is proposed to partly address this problem. However, GAC-schema is a general framework and does not address how to deal with special constraints such as linear arithmetic constraints efficiently.

3 Bounds consistency on linear constraints

The first part of this section introduces the specialization of n-ary CSPs to linear arithmetic constraints and defines bounds-consistency on them. The second part presents bounds-consistency algorithms and their associated complexity analysis. We denote the set of integers by Z.

3.1 Linear constraint and bounds-consistency

Definition 4. A linear arithmetic constraint $c_{\{x_1,\cdots,x_n\}}$ *is of the form*

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n \diamond\ b$$

$$a_i, b \in Z \qquad \diamond \in \{=, \leq\}.$$

where *vars*(c) and $|c|$ is used to denote the set and the number of variables that occur in c respectively. A linear constraint system representing a n-ary CSP is one where all constraints are linear arithmetic constraint and all domains contain only integers. Other linear arithmetic constraints with $(<, >, \geq)$ can be rewritten in the above form.

Essentially, the problem of enforcing n-ary arc consistency is related to that of finding all solutions satisfying the given linear constraint. This may be quite expensive. One well known way to reduce this cost is to relax domains of the variables so that they form a continuous real interval bounded by the maximum and minimum values of the corresponding domains. Since variables can now take real values and are no longer discrete, it is easy to make the constraint arc consistent. We now make this precise. First, we introduce some basic interval arithmetic operations [20] which will simplify our presentation.

Assume that each variable x is associated with an interval $[l, u]$. We use $[x]$ and $\langle x\rangle$ to denote two kinds different kinds of operations: an interval operation; and a literal operation on x respectively. Let l, u denote the interval associated with x, we use the following notation:

$$[x] = [l, u] \qquad\qquad \langle x\rangle = \begin{pmatrix} l \\ u \end{pmatrix}$$

Given $[x] = [l_1, u_1]$ and $[y] = [l_2, u_2]$, the interval operations are defined in the usual fashion:

$$[x] + [y] = [l_1 + l_2, u_1 + u_2],$$

$$[x] - [y] = [l_1 - u_2, u_1 - l_2],$$

$$[x] - a = [l_1 - a, u_1 - a],$$

$$a[x] = \begin{cases} [al_1, au_1], a > 0 \\ [au_1, al_1], a < 0, \end{cases}$$

$$[x] \cap [y] = \big[\mathit{max}(l_1, l_2), \mathit{min}(u_1, u_2)\big]\,.$$

The literal operations unlike the interval operations are defined as a pairwise tuple operation, which differs in subtraction from the interval counterpart:

$$\langle x\rangle \pm \langle y\rangle = \begin{pmatrix} l_1 \pm l_2 \\ u_1 \pm u_2 \end{pmatrix}.$$

We will for convenience also overload the [ ] and $\langle\ \rangle$ notation. We use $\langle[x]\rangle$ to mean a substitution of the literal operation for the interval operation.

The following example is now used to motivate the use of interval reasoning for consistency,

$$3x - 4y = 0, [x] = [y] = [1, 10].$$

Clearly, y cannot take the value 10 no matter what value x takes. More precisely, given any value of x in [1,10], y can only take a value in [3/4, 30/4]. So the set of valid values of y with respect to the above constraint is [3/4, 30/4] $\cap$ [1,10]=[3/4, 30/4]. The above process to remove invalid values can be formalized as follows.

Definition 5. *The* projection function π_i *of a constraint* c *on* x_i *is*

$$\pi_i(c) = \frac{-1}{a_i}(a_1x_1 + \cdots + a_{i-1}x_{i-1} + a_{i+1}x_{i+1} + \cdots + a_nx_n - b).$$

Given intervals on all the variables, we can define the interval version of the projection of c *on* x_i *as:*

$$\Pi_i(c) = \frac{-1}{a_i}[a_1[x_1] + \cdots + a_n[x_n] - b].$$

We call $\Pi_i(c)$ *the* natural interval extension *of* $\pi_i(c)$.

We now define the function $Proj_i(c)$ as follows:

$$Proj_i(c) = \begin{cases} \Pi_i(c) & \text{if } \diamond' \text{ is } = \\ [-\infty, Ub(\Pi_i(c))] & \text{if } \diamond' \text{ is } \leq \\ [Lb(\Pi_i(c)), +\infty] & \text{if } \diamond' \text{ is } \geq \end{cases}$$

where

$$\diamond' = \begin{cases} \geq \text{ if } a_i \text{ is negative and } \diamond \text{ is } \leq \\ \diamond \text{ otherwise} \end{cases}$$

and $Ub([l, u]) = u, Lb([l, u]) = l$.

As a consequence of the intermediate value theorem from calculus, we have the following property.

Property 1. Given a constraint c with initial domains $([x_1], \cdots, [x_n])$, the constraint c is arc consistent with respect to the new domain $([x_1] \cap Proj_1(c), \cdots, [x_n] \cap Proj_n(c))$.

The relaxation of the domain of a variable from discrete to a continuous real interval allows efficient arc consistency enforcement for a single linear constraint in the time needed for computing n operations of $Proj_i(c)$. However for a system of constraints, this process may not terminate [13].

We now define bounds-consistency. Instead of using the real interval relaxation, we restrict the interval to the *Z-interval* whose upper bound and lower bound are integers. The Z-interval representation of a set $S \subset \mathcal{R}$ is $\Box S = [\lceil u \rceil, \lfloor v \rfloor]$ where u and v is the minimum and maximum real values in S respectively.

Definition 6. *A constraint c is* bounds-consistent *with respect to* $(\Box D_{x_1}, \cdots, \Box D_{x_n})$ *iff* $\forall x_i \in \mathrm{vars}(c)\quad \Box D_{x_1} \subseteq \Box Proj_i(c_i)$. *A linear constraint system (N, D, C) is* bounds-consistent *with respect to* $(\Box D_1, \cdots, \Box D_m)$ *iff every* $c_i \in C$ *is bounds-consistent.*

3.2 Bounds consistency algorithm and its complexity

Although the definition of bounds-consistency holds for n-ary linear constraints, it fits well in an AC-3 style computation framework which is normally only used for binary constraints. We now describe a AC-3 like algorithm to achieve bounds-consistency on a system of linear constraints. We chose this presentation for two reasons. It is a simple and natural algorithm and for that reason would be similar to general propagation and filtering based algorithms as well. Unlike AC-3, the basic unit of manipulation here is a single constraint. A queue is employed to hold those constraints needing update when the domain of some of its variables is changed. The algorithm BC is listed in figure 1. The difference between BC and AC-3 is that the REVISE procedure is specialized for bounds-consistency and linear constraints.

We point out that the operation in line 1 of BC is different from the *narrowing operation* [3] in that the Z-interval representation performs inward rounding while for continuous intervals represented by floating point numbers it an outward rounding operation. Note that the narrowing operation on c_j defined by REVISE is no longer idempotent given inward rounding.

Lemma 1. *Given a linear constraint system* (N, D, C), *the worst case time complexity of algorithm BC is* $\mathcal{O}(en^3d)$

Proof. The worst case complexity of BC depends on the number of constraints ever entering the Queue Q. A constraint c enters Q iff some value in some domain involved in c is deleted. For each variable $x_i \in N$, assume it appears in k_i constraints. In total, we have md values in the system where m is the number of variables in C. Thus the number of constraints ever entering Q is at most $\sum_{i=1}^{m} d \cdot k_i$. Let α be $\sum_{i=1}^{m} k_i$. A loose estimate of k_i can be simply e which means the variable can appear in any constraint in the system. However, a relatively tighter estimation for α is as follows. Consider the bipartite graph $G_{m,e}$ with vertices sets N and C. There is an edge between $x_i \in N$ and $c_j \in C$

```
Algorithm BC
begin
  $Q \leftarrow \{c_i | c_i \in C\}$;
  while ($Q$ not empty)
  begin
    select and delete $c_i$ from $Q$;
    REVISE($c_i, Q$);
  end
end
procedure REVISE($c_j, Q$)
begin
  for each $x_i \in vars\ (c_j)$
  begin
    if $[x_i] \not\subseteq \Box Proj_i(c_j)$
      begin
1.      $[x_i] \leftarrow [x_i] \cap \Box Proj_i(c_j)$;
2.      $Q \leftarrow \{c_k \in C \mid x_i \in vars\ (c_k)\}$
      end
  end
end
```

Fig. 1. Algorithm BC

iff x_i appears in c_j. α is exactly the number of edges of $G_{m,e}$. Since the degree of c_j is not more than n we have that the number of edges in $G_{m,e}$ is less than ne, that is $\alpha \leq ne$. The complexity of procedure REVISE is at most n^2. Therefore the complexity of BC is $\mathcal{O}(en^3d)$. □

The naive algorithm can be improved by making REVISE more efficient using the following result.

Proposition 1. *Given an n-ary linear arithmetic constraint system* (N, D, C)*, bounds-consistency can be achieved in time* $\mathcal{O}(en^2d)$

Proof. To improve the efficiency of BC, one way is to make REVISE faster. Let constraint c_j be

$$a_{j_1}x_1 + a_{j_2}x_2 + \cdots + a_{j_n}x_n \diamond b_j.$$

Let

$$f_j = a_{j_1}x_1 + a_{j_2}x_2 + \cdots + a_{j_n}x_n - b_j$$

Let F_j be the natural interval extension of f_j. Now, for any $x_i \in c_j$

$$\Pi_i(c_j) = -\frac{1}{a_{j_i}}[\langle F_j \rangle - \langle a_{j_i}[x_i] \rangle].$$

since we have that

$$\begin{aligned} \langle F_j \rangle - \langle a_{j_i}[x_i] \rangle &= \langle [a_{j_1}[x_1] + \cdots a_{j_i}[x_i] + \cdots a_{j_n}[x_n] - b_j \rangle - \langle a_{j_i}[x_i] \rangle \\ &= a_{j_1}[x_1] + \cdots + a_{j_{i-1}}[x_{i-1}] + a_{j_{i+1}}[x_{i+1}] + \cdots + a_{j_n}[x_n] - b_j \end{aligned}$$

Note the f_j is not a projection function and the use of the literal $\langle\rangle$ operations in $\Pi_i(c_j)$. According to the definition of $Proj_i(c_j)$, REVISE can be implemented in linear time of n. So, the BC algorithm can be implemented in time of $\mathcal{O}(en^2d)$. $\square$

4 Linear inequalities and monotonic constraint

We will now consider a system of linear inequalities. For a system of linear inequalities, we have the following result without any relaxation of the Z domain to Z-intervals.

Proposition 2. *Given an n-ary CSP (N, D, C) which consists only of linear inequalities, it will be arc consistent after bounds-consistency is enforced on it.*

Proof. Assume CSP (N, D, C) is bounds-consistent. Now we show that any constraint c_j is arc consistent with respect to D. Consider any variable x_i, $x_i \in vars(c_j)$, and any value v, $v \in D_i$. Let l and g be the least and greatest integers in D_i. Without loss of generality, assume that $a_i > 0$, we have $x_i \leq \pi_i$. Because the system is bounds-consistent, we have $[l, g] \subseteq \square Proj_i(c_j)$, which means that $v \leq g \leq Ub(Proj_i(c_j))$ where $Ub(Proj_i(c_j))$ is obtained by letting $x_k = v_k, k : 1 \dots n, k \neq i$ where v_k is either the lower bounds or the upper bounds of D_k depending on the interval operation. So, $(v_1, \cdots, v_{i-1}, v, v_{i+1}, \cdots, v_n)$ satisfies c_j. Similarly, when $a_i < 0$, we can prove v is part of a solution of c_j. $\square$

It follows immediately that a system of linear inequalities can be made arc consistent in worst case time complexity of $\mathcal{O}(en^2d)$.

This result can be generalized to a bigger class of n-ary constraints, the n-ary monotonic constraints. We begin by recalling the definition of binary monotonic constraint in [28]. From now on, we assume that all the domains D_i are finite and have a total ordering.

Definition 7. *[28] Given a binary CSP (N, D, C), a constraint $c \in C$ is* mono-tonic *with respect to domain $TD = \cup_{i=1}^{m} D_i$ iff there exists a total ordering on TD such that for all values $v, w \in TD$ and $c(v, w)$ implies $c(v', w')$ for all $v' \leq v$ and $w' \geq w$.*

An example of an arithmetic constraint which is monotonic under this definition is $x \leq y, [x] = [y] = [1, 10]$. However, with this definition, the linear inequality $x + y \leq 10, [x] = [y] = [1, 10]$ is *not* a monotonic constraint. For example, consider $x = 5, y = 5$ as an a valid pair, then $x' = 5, y' = 6$ is not consistent using the natural ordering. There is no total ordering on TD which makes this constraint monotonic.

However applying algorithm BC, a binary system of both kinds of constraints can be made arc consistent in time $\mathcal{O}$(ed). Thus we see that this definition of monotonicity is stronger than necessary and does not fully exploit the special properties of inequalities which give more efficient arc consistency algorithms. We now give the following generalization of binary monotonic constraint which remedies this problem by relaxing the total ordering requirement on the union of all the domains.

Definition 8. *Given a binary CSP (N, D, C), a constraint $c_{\{i,j\}} \in C$ is* monotonic *iff there exists a total ordering on D_i and D_j respectively such that $\forall v \in D_i, \forall w \in D_j$ $c(v, w)$ implies $c(v', w')$ for all $v' \leq v$ and $w' \geq w$.*

Consider again the example, $x + y \leq 10, [x] = [y] = [1, 10]$. This is now monotonic. A possible ordering is the natural one on x, and on y we have the reverse ordering. Now we have a natural extension of monotonicity to n-ary general constraints.

Definition 9. *Given an n-ary CSP (N, D, C), a constraint $c_X \in C$ is* monotonic *with respect to variable $i \in X$ iff there exists a total ordering on D_1 to D_n respectively such that $\forall v \in D_i, \forall v_j \in D_j$ $c_X(v_1, \cdots, v_{i-1}, v, v_{i+1}, \cdots, v_n)$ implies $c(v'_1, \cdots, v'_{i-1}, v', v'_{i+1}, \cdots, v'_n)$ for all $v' \leq v$ and $v'_j \geq v_j$ for $j \in X, j \neq i$. A constraint $c_X \in C$ is* monotonic *iff c_X is monotonic with respect to all variables of X.*

It is easy to verify that any n-ary linear arithmetic inequality is monotonic. Another example of a monotonic constraint is, $x * y \leq z, D_x = D_y = D_z = \{1, \ldots, 100\}$. For finite domain constraints, our definition of monotonic constraints is more general than the monotonic functions defined in [11].

In order to achieve arc consistency on monotonic constraints, the REVISE in algorithm BC should be modified as in Figure 2. It is important to note that in the new algorithm, an explicit projection function is not required. At the initialization phase of BC, for any constraint c and $i \in vars(c)$, we explicitly store the particular ordering of each domain involved which makes c monotonic with respect to i.

```
Procedure REVISE($c_j, Q$)
begin
  for each $x_i \in vars\ (c_j)$
  begin
    $\forall j, v_j \leftarrow$ the greatest value in $D_j$ wrt $x_i$
    DELETE = 0;
1.  while (not c($v_1, \cdots, v_n$))
      begin
        remove $v_i$ from $D_i$;
        DELETE = 1;
        $v_i \leftarrow$ the greatest value in $D_i$
      end
    if DELETE
      $Q \leftarrow \{c_k \in C \mid x_i \in vars\ (c_k)\}$
  end
end
```

Fig. 2. REVISE for monotonic constraint

Proposition 3. *Given a CSP (N, D, C) which contains only monotonic constraints, it can be made arc consistent in time complexity of $O(en^3d)$ if the complexity of evaluating $c(v_1, \cdots, v_n)$ is $O(n)$.*

The sketch of the proof is as follows. In a similar fashion to Proposition 2, we can show that arc consistency can be achieved on monotonic constraints. The complexity of the algorithm depends on the execution times of line 1 in the REVISE of Figure 2. If we expand one execution of the algorithm according to line 1, executions of line 1 can be separated into two groups. One group contains executions without any value removed and the other group contains executions with at least one value removed. Because REVISE can be executed at most n^2ed times, the complexity of executions of the first group is n^3ed according to the linear time evaluation of c. As for the second group, we cluster the computation around variables. Now the total computation is

$$\sum_{i=1}^{m} n \cdot (d_{i,1} + \cdots + d_{i,k}) \leq \sum_{i=1}^{m} n \cdot d \leq mnd$$

where $d_{i,l}(l : 1..k)$ denotes the number of elements removed from D_i in some execution of the while loop in line 1 on i. Because $m \leq ne$, the complexity of the second group will be smaller than the first group and thus the complexity of the algorithm is $\mathcal{O}(en^3d)$. $\square$

We remark that, as in proposition 1, by using the special semantics of monotonic constraint, it may be possible to decrease the complexity of the arc consistency algorithm by a factor n.

We now would like to briefly discuss how to embed the monotonic arc consistency algorithm into a general algorithm. AC-5 [28] does not discuss how this is to be done and leaves it as an implementation detail. The AC-6 algorithm is a suitable candidate for this. To simplify the discussion, we will illustrate the idea using a binary monotonic $c_{\{x,y\}}$ given in Figure 3.

In the initialization phase of AC-6 for $c_{\{x,y\}}$, we only need the least value in x and greatest value in y. The ordering used here gives a as the least value in x and g as the greatest value in y. In the implementation, we can easily associate the values a and g with the revision process for $c_{\{x,y\}}$. Now, any deletion of values of b, c, e, or f by other constraints will not invoke the revision of constraint $c_{\{x,y\}}$. Only when a (or g) is removed will monotonic constraint revision be invoked. After the monotonic revision process finishes, it will associate the revision process again to the new least (or greatest) values left. This approach conforms to the *lazy* principle behind AC-6.

5 Linear equations

We now consider n-ary CSPs where the constraints are linear equations. The importance of this section is that the complexity results are very different from the $\leq$ case. In the equation case when the domains are considered to be discrete,

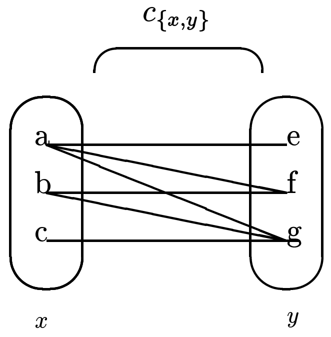

Fig. 3. A monotonic constraints embedded in AC-6

bounds-consistency does not imply arc consistency. It is only if we relax the domains to be Z-intervals that bounds-consistency implies arc consistency.

Unfortunately, the problem of enforcing arc consistency on a single linear equation is a very hard problem. Recall that arc consistency in the n-ary case means that we need to show that single constraints are satisfiable by themselves. Consider the one-line integer programming problem: Is there a 0-1 n-vector x such that

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b$$

where $b, a_1, \cdots, a_n$ are given positive integers? The above problem is NP-complete [23]. Obviously, enforcing arc consistency on a system of linear equations is also NP-complete. It is also immediate that enforcing arc consistency on any single arbitrary n-ary constraint is NP-complete in the worst case.

This observation highlights the computational difficulty with n-ary constraints and arc consistency. Arc consistency on linear inequalities (also monotonic constraints) is tractable, however generalizing to arbitrary linear constraints such as linear equations makes arc consistency intractable. This distinction can also be viewed as the difference in arc consistency between different representations. One can choose to represent linear equations as two inequalities per equation, eg. $exp = b$ as $exp \leq b, exp \geq b$. In the continuous case, arc consistency on the original and double inequality representation gives the same resulting domains in the same time complexity. In the discrete case, the two inequality representation can be made arc consistent as in Section 4. It does not however make the original equations arc consistent since arc consistency treats each inequality separately.

Now consider relational consistency as defined in [26]. On a system of linear inequalities, relational arc consistency can be achieved in polynomial time, however enforcing relational path consistency is NP-complete.

6 Discussion and Conclusion

We now discuss the relationship of our work with that in the continuous domain. A substantial body of work in n-ary constraints comes from the continuous domain rather than the discrete domain. The early work [11, 22] focused mainly on issues of correctness, convergence, searching strategy, etc. In more recent work the emphasis is on using numerical methods such as Newton methods [2] and Aitken acceleration [15] to speedup convergence. Our definition of bounds-consistency is similar to *arc B consistency* [16] and *interval consistency* [2, 9] but differs in that bound-consistency uses an inward rounding operation. The time complexity of filtering algorithms in the continuous domain, on the other hand, is usually not treated for the following reasons. Firstly for real/rational intervals, the *interval Waltz filtering* algorithm may not terminate given arbitrary linear constraints [8]. Secondly for floating point intervals, the domain is huge and thus the worst case time complexity may not be of practical relevance and efficiency is gained not so much by reducing the time complexity, but by faster convergence using numerical methods. In [16], existing complexity results from general discrete arc consistency algorithm are used to bound their filtering algorithms. Thus, the work in the continuous case does not directly help in getting more efficient algorithms and their resulting time complexity analysis in the discrete case.

n-ary discrete constraints, including integer linear constraints [21], are widely used for modelling and solving many problems in systems for constraint programming using finite domain solvers [7, 12, 27]. Such solvers use various techniques based on the propagation of bounds for arithmetic constraints [14]. The use of bounds based propagation techniques is not new and originates as early as in 1978 [14]. However, the efficiency and level of consistency of such techniques is not studied and described in detail. In this paper, we address the question of what level of consistency can be achieved efficiently on n-ary linear constraints. The observation from Section 5 shows that arc consistency on n-ary linear equations is not tractable. We carefully introduce and formalize the notions of bounds-consistency in the context of discrete CSP. It is shown that arc consistency for linear inequalities system can be achieved with a simple AC-like algorithm in time complexity $\mathcal{O}(en^3d)$. Where an efficient implementation of REVISE is possible as is the case with the projection of linear inequalities, the time complexity is improved to $\mathcal{O}(en^2d)$.

Given that arc consistency on a single n-ary constraint can be NP-complete, we identify a general class of monotonic constraints (which need not be linear) for which arc consistency can be efficiently enforced. Monotonic constraints are actually a special case of *row convex constraints* [26]. [26] presents an algorithm achieving *relational path consistency* for row convex constraints but it behaves exponentially even for a system of two n-ary monotonic constraints (since it is an NP-complete problem).

The work in this paper also extends the results in [28] and complements the GAC-schema [5].

Some open questions suggested by the results here are the following. What are other general classes of n-ary constraints for which enforcing arc consistency is efficient. What is the optimal time complexity for arc consistency on linear inequalities and monotonic constraints?

References

1. F. Bacchus and P. van Beek, "On the conversion between non-binary and binary constraint satisfaction problems", *Proceedings of AAAI-98*, Madison, WI, 1998
2. F. Benhamou, D. McAllester, and P. van Hentenryck, "CLP(*intervals*) Revisited", *Proceedings of 1994 International Symposium on Logic Programming*, 124–138, 1994
3. F. Benhamou and W. Older, "Applying Interval Arithmetic to Real Integer and Boolean Constraints", *Journal of Logic Programming* 32(1), 1997
4. C. Bessiere, "Arc-consistency and arc-consistency again", *Artificial Intelligence* 65:179–190, 1994
5. C. Bessiere and J. Regin, "Arc consistency for general constraint networks: preliminary results", *Proceedings of IJCAI-97*, Nagoya, Japan, 1997
6. C. Bessiere and J. Regin, "MAC and combined heuristics: two reasons to forsake FC(and CBJ?) on hard problems", *Proceedings of Principles and Practice of Constraint Programming*, Cambridge, MA. 61–75, 1996
7. P. Codognet and D. Diaz, "Compiling Constraints in CLP(FD)", *Journal of Logic Programming* 27(3), 185–226, 1996
8. E. Davis Constraint Propagation with Interval Labels *Artificial Intelligence* 32, 281–331, 1987
9. R. Dechter, I. Meiri and J. Pearl Temporal constraint networks *Artificial Intelligence* 49, 61–95, 1992
10. R. Dechter and J. Pearl, "Tree clustering for constraint networks", *Artificial Intelligence* 38:353–366, 1989
11. E. Hyvonen Constraint reasoning based on interval arithmetic: the tolerance propagation approach *Artificial Intelligence* 58, 71–112, 1992
12. ILOG, *ILOG SOLVER Reference Manual*
13. Joxan Jaffar, Michael J. Maher, Peter J. Stuckey and Roland H.C. Yap Beyond Finite Domains *PPCP'94: Proceedings of the Second Workshop on Principles and Practice of Constraint Programming*, 1994, 86–94
14. J. Lauriere, "A language and a program for stating and solving combinatorial problems", *Artificial Intelligence* 10:29–127, 1978
15. Y. Lebbah and O. Lhomme Acceleration methods for numeric CSPs *Proc. of AAAI-98* 1998
16. Lhomme, Olivier, Consistency Techniques for Numeric CSPs, *Proceedings of IJCAI-93*,Chambery,France,232–238, 1993
17. A. K. Mackworth, "Consistency in Networks of Relations", *Artificial Intelligence* 8(1):118–126, 1977
18. A. K. Mackworth, "On reading sketch maps", *Proceedings of IJCAI-77*, 598–606, Cambridge MA, 1977
19. R. Mohr and G. Masini , "Good old discrete relaxation", *Proceedings of ECAI-88*, 651–656, Munchen, FRG, 1988
20. R. E. Moore, *Interval Analysis*, Prentice Hall, 1966
21. G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*, New York, Wiley, 1988
22. Older, W. and Vellino,A. Constraint Arithmetic on Real Intervals *Constraint Logic Programming:Selected Research*, Benhamou, F. and Colmerauer, A. (eds.), 175–195,1993
23. C. H. Papadimitriou, "On the complexity of integer programming", *J. of the ACM* 28(4):765–768, 1981

24. J. C. Regin, “Generalized arc consistency for global cardinality constraint”, *Proceedings of AAAI-96*, 209–215, Portland, OR, 1996
25. F. Rossi, C. Petrie, and V. Dhar, “On the equivalence of constraint satisfaction problems”, *Proceedings of the 9th European Conference on Artificial Intelligence*, 550–556, Stockholm, Sweden, 1990
26. P. van Beek and R. Dechter, “On the minimality and global consistency of row-convex constraint networks”, *Journal of the ACM* 42(3):543–561, 1995
27. P. van Hentenryck, *Constraint Satisfaction and Logic Programming*, MIT Press, Cambridge, 1989
28. P. van Hentenryck, Y. Deville, and C. M. Teng, “A Generic Arc-Consistency Algorithm and its Specializations”, *Artif. Int.* 58(1992):291–321, 1992
29. P. van Hentenryck, L. Michel, and Y. Deville, *Numerica: A Modeling Language for Global Optimization*, MIT Press, Cambridge
30. D. L. Waltz, “Generating semantic descriptions from drawings of scenes with shadows”, *MAC-AI-TR-217*, MIT, 1972