Towards Automatic Generation and Evaluation of Implied Constraints

Brahim Hnich[1], Julian Richardson[2], and Pierre Flener[1]
[1]Dept. of Information Science
Division of Computer Science
Uppsala University
S–751 20 Uppsala, Sweden
{ Brahim.Hnich, Pierre.Flener } @dis.uu.se
[2]Dept. of Computing and Electrical Engineering
Heriot-Watt University
EH 14 4AS Edinburgh, Scotland, UK
julianr@cee.hw.ac.uk

1 Introduction

Production planning subject to demand and resource availability so that profit is maximized, air traffic control subject to safety protocols so that flight times are minimized, transportation scheduling subject to initial and final location of the goods and the transportation resources so that the delivery time and fuel expenses are minimized, and many other real life examples can be stated as optimization Constraint Satisfaction Problems (CSP).

A CSP is a 4-tuple $\langle Vars, Doms, Cons, Obj\rangle$, where each variable V_i in $Vars$ has domain D_i in $Doms$. Each k-ary constraint C in $Cons$ states a relation between k variables in $Vars$. Obj is an optional objective function over the variables in $Vars$. Solving a CSP requires assigning values to the variables from their corresponding domains such that all the constraints in $Cons$ are satisfied and the objective function is minimium. It is in general an *NP-complete* problem [30]. We here restrict ourselves to decision CSPs, where there is no objective function.

The variables may have different types: integer, real, boolean, set, list, etc. The domains can either be finite or infinite. The constraints may also have different forms such as being arithmetic (linear and nonlinear), symbolic, set operations, list operations, etc. There are different methods for different instances of the general CSP in the litterarture. In operation research many methods were developed to deal with CSP. *Linear Programming* (LP) [10] allows a linear objective function subject to a set of linear constraints over real variables that are nonnegative. Polynomial time robust solvers are available that can solve large scale linear programs. A generalization of LP is *Integer Linear Programming* (ILP) [16] where variables range over integers. However, this makes the problem NP-complete. A Further generalization is when the objective function and the constraints are allowed to be nonlinear. *Constraint (Logic) Programming* (CP) [24] is an orthogonal paradigm to I(L)P to solve CSP. Many different solvers have been proposed. Finite Domain (FD) solvers handle linear/nonlinear and symbolic constraints and the variables may take discrete values [4, 40]. Real solvers like in [21] efficiently handle linear constraints and uses lazy evaluation for the nonlinear ones. In [17, 36], variables are allowed to range over finite sets and constraints may have set operations.

Restricting the type of the variables, the domains, and the forms of the constraints can help developing more specialized methods to tackle those subclasses. For instance, if the variables are integers and the constraints are conjunctive linear inequalities, then there has been appropriate tools developed especially to handle that such as ILP. If the domains of the variables are finite, then consistency techniques combined with a search algorithm can be viewed as a special method that can handle effectively some problems, especially if the constraints are nonlinear. We call *restricted CSP* any subclass of the general CSP, where the domain type and/or the variable type and/or the form of the constraints has been restricted. Restricted CSPs are less expressive than the general ones, hence special methods that take advantage of those restrictions are in general more efficient. The consequences of this are the following:

- The restricted language is more efficient.
- The restricted language is less expressive.
- The restricted language will either not be able to handle expressions that do not obey the restrictions or handle them very inefficiently.

Given a restricted CSP, one of the major goals is to extend the expressivness without decreasing the efficiency, or to improve the efficiency without loosing expressivness, or both, whenever possible. One way to achieve this is to reason on the constraints themselves. For example, in the case of ILP, one can extend the expressivness by allowing nonlinear constraints (*alien constraints*) to be formulated, but then those nonlinear constraints should be mapped to linear ones for which the ILP methods would know how to handle very efficiently. Another example is with constraint programs over finite domains; one can add *redundant* constraints —which are implied by the constraints of the original problem— to the program which would help achieve more pruning and hence increase the efficiency.

Adding implied constraints –which are redundant constraints– to the original problem may lead to a significant increase in the performance and may extend the expressivness of the language. In the case of finite domain solvers it may help pruning more inconsistent values from the variables domains, and hence yielding a smaller search space. So far, researchers have been adding implied constraints to the problems manually, or by using very highly specialized methods for deriving a particular class of implied constraints such as the valid cuts. We will focus on CSPs over finite domains. Furthermore, the generation of implied constraints for constraint programs over finite domains can be done at compile-time or run-time. Run-time generation can be done prior to the start of the search (*pre-search generation*) as well during the search itself (*within-search generation*.) We will focus on compile time and pre-search generation.

Driven by the importance of implied constraints being added to the original problem, we are aiming at the following:

- A classification of the implied constraints
- Since implied constraints are logically implied by the set of initial constraints, then *Automated Theorem Proving* (ATP) technologies will be explored. We will experience with and evaluate different existing systems that have a potential of generating implied constraints. Namely, we will study and evaluate the following theorem provers in a black-box approach:
  - Press system: heuristics-based theorem prover that deals with linear and nonlinear constraints alike.
  - clp(q,r) solver: employs a decision algorithm for linear inequalities which derives implied constraints and handles linear (dis-)equations.
  - Otter: brute-force approach...
- Compare the different theorem provers and propose a general method that would combine the pros and discard the cons of the existing systems. This will result in a tool that automatically generates implied constraints.
- As the set of implied constraints generated may be huge, we will try to devise some methods to choose "useful" implied constraints to add to the original problem and discard the "un-useful" ones. Hence we will try to come out with some evaluation methods for the generated implied constraints. This direction of research may reveal some results that can be incorporated in the generation step, so as to generate only "useful" implied constraints.
- develop methods that analyze the problems and decide if we need to add implied constraints or not to the initial problem.

This report is organized as follows: in Section 2 we will present some motivating examples and background knowledge in Section 3. Section 4 will describe related work while in Section 5 we will present our results on the classification and generation of implied constraints. In Section 6, the focus will be on the evaluation of implied constraints in the finite domain case where consistency methods are employed. We will elaborate on future activities in Section 7 and conclude in Section 8.

2 Motivation

Constraints over Finite Domains

If the domains of the variables are finite, then the CSP can be modeled as a constraint program and solved by a constraint solver over finite domains. Due to the the propagation algorithms employed by the solver, some level of consistency is maintained. Maintaining consistency leads to the pruning of inconsistent values from the domains of the variables, and hence will produce a smaller search tree. More pruning can be achieved, if some *redundant constraints* (constraints that logically follow from the initial set of the constraints) are added to the problem as shown in [35, 18, 32, 37].

Example 2.1 For the following assume we are using the clp(FD) solver of Sicstus Prolog [4] as black-box, which maintains *interval consistency* for the variables which have interval domains. Suppose we have the following toy problem:

$$\begin{gathered} Vars = \{A, B, C\} \\ Dom = \{0..100, 0..100, 0..100\} \\ Cons = \{\ A + B + C = 178, 2A - B + 4C = 99, B + C = A\} \end{gathered} \tag{1}$$

If we call the CLP(FD) solver, then the domains of the variables will be as follows:

$$\begin{gathered} A \in 59..99 \\ B \in 59..99 \\ C \in 0..20 \end{gathered} \tag{2}$$

But, by some simple algebraic manipulation we can derive the following:

$$A + B + C = 178 \wedge B + C = A \rightarrow B + C = 89 \tag{3}$$

So, we add this constraint to our initial problem, yielding the following:

$$\begin{gathered} Vars = \{A, B, C\} \\ Dom = \{0..100, 0..100, 0..100\} \\ Cons = \{\ A + B + C = 178, 2A - B + 4C = 99, B + C = A, B + C = 89\} \end{gathered} \tag{4}$$

By calling the CLP(FD) solver,the domains of the variables will be as follows (which is clearly better than the original model):

$$\begin{gathered} A \in 84..94 \\ B \in 84..89 \\ C \in 0..5 \end{gathered} \tag{5}$$

We can also derive the following:

$$A + B + C = 178 \wedge 2A - B + 4C = 99 \rightarrow 3A + 5C = 277 \tag{6}$$

Now, adding this implied constraint to our initial problem will yield the following:

$$\begin{aligned} Vars &= \{A, B, C\} \\ Dom &= \{0..100, 0..100, 0..100\} \\ Cons &= \{\ A + B + C = 178, 2A - B + 4C = 99, B + C = A, 3A + 5C = 277\} \end{aligned} \tag{7}$$

By maintaining the same level of consistency, the domains of the variables will be as follows (note that no labeling is needed as all the domains are singleton):

$$\begin{aligned} A &= 89 \\ B &= 87 \\ C &= 2 \end{aligned} \tag{8}$$

From the example, we can see that adding implied constraints to our initial set of constraints helps pruning more the domains of the variables and thus achieve a higher level of consistency. However, not all implied constraints help achieving more pruning. Here is an example of such redundant constraint:

$$\begin{aligned} Vars &= \{A, B, C, D\} \\ Dom &= \{0..100, 0..100, 0..100, 0..100\} \\ Cons &= \{\ A + B - C = 100, A + D = 50\} \end{aligned} \tag{9}$$

Maintaining consistency will yield the following domains for the variables:

$$\begin{aligned} A &\in 0..50 \\ B &\in 50..100 \\ C &\in 0..50 \\ D &\in 0..25 \end{aligned} \tag{10}$$

We can derive the following implied constraint:

$$A + B - C = 100 \wedge A + D = 50 \rightarrow B - 2D - C = 50 \tag{11}$$

However, adding this constraint to the initial problem will not lead to any change in the variables domains after maintaining the same level of consistency.

Example 2.2 Given the following CSP, where all variables have domain 1..9,

$$\begin{aligned} Vars &= \{A, B, C, D, E, F, G, H, I\} \\ Cons &= \{A/B * C + D/E * F + G/H * I = 1, \\ A * E * F &\geq D * B * C, D * H * I \geq G * E * F\} \end{aligned} \tag{12}$$

We can derive the implied constraints $3 * A \geq B * C \wedge 3 * G \leq H * I$. If we run the original problem on Sicstus CLP(FD) then it takes 17240 msec and 31051 backtracks, while if we add the implied constraints the run-time reduces to 13640 msec and the backtracks to 24827. Note that the derivation time of the implied constraint is not measured here. It will be worthwhile to generate such an implied constraint if it takes less than 17240 msec minus 13640 msec.

Constraints over Real Domains

If the domains of the variables are rational or real valued, then constraint solvers over rational domain may be employed. However, such systems, such as clp(q,r) [21] are mainly designed for dealing with linear constraints and use a lazy evaluation approach for the nonlinear case, i.e., it collects the nonlinear constraints with the hope that through the addition of further linear constraints they might get simple enough to solve, otherwise a solution is found and checked for the nonlinear constraints . On the other hand, if some redundant linear constraints –that can be implied by the linear and nonlinear constraints – are added to the original set of constraints may make such solvers cope in a better way with the nonlinear constraints.

Example 2.3 For the following assume we are using the clp(q,r) solver of Sicstus Prolog [21] as black-box. Suppose we have the following toy problem (here we assume that all variables are real valued):

$$\begin{gathered} Vars = \{A, B, C\} \\ Cons = \{A + B + C \leq 6, A^2 > 1, A \geq 0, B \geq 0, C \geq 0\} \end{gathered} \tag{13}$$

The clp(q,r) will delay the solving of the constraint $A^2 > 1$. But, we can infer from the constraints the *linear* constraint $B + C < 5$ which can be handled efficiently by clp(q,r), since $A^2 > 1 \wedge A \geq 0 \rightarrow A > 1$ and $A > 1 \wedge A + B + C \leq 6 \rightarrow B + C < 5$.

3 Background

3.1 Compile Time Versus Runtime Generation of Implied Constraints

Our focus in this work is on CSPs where the domains are finite with no objective function. In what follows we will use CSP to denote this subclass. We here define compile-time and run-time generation of implied constraints. If the generation is at compile-time, then the generation time will be irrelevant while if the generation is at run-time, then it will be counted as part of the time spent to actually solve the problem. We start first by some definition.

Definition 3.1 *Implied constraint:*
Given a set C of constraints, an implied constraint *is any formula θ such that $C \vdash \theta$.*

Definition 3.2 *Equivalent sets of constraints:*
Given two sets C_1 and C_2 of constraints, C_1 and C_2 are equivalent *iff $C_1 \vdash C_2 \wedge C_2 \vdash C_1$.*

Definition 3.3 *Redundant Constraint:*
Given a $CSP = \langle V, D, C\rangle$ and $CSP_R = \langle V \cup V', D \cup D', C \cup \{R\}\rangle$, where R is a

constraint not appearing in C, and V' is the set of extra variables (may be empty) that are constrained by R and having corresponding (may be empty) domains in D'. R is said to be redundant*, iff for the set of solutions of CSP denoted by $sol(CSP)$ and for the projection of the set of solution of CSP_R over the set of variables V denoted by $\sigma_V sol(CSP_R)$ we have $sol(CSP) = \sigma_V sol(CSP_R)$.*

It follows from the definitions that all implied constraints are redundant and if a subset of C is equivalent to C' then all the constraints in C' are redundant.

Solving a CSP requires that the set of variables V, their corresponding set of domains D, and the set of constraints C should be available. Once, all the information about the CSP is available, any kind of processing or solving is a run-time process because if it was a compile-time process then one can use a different approach or even the same solver to actually solve the problem and then call the solver with a solved CSP that gives an answer in constant time, which will make it look as if we have a constant time algorithm for solving an NP-complete problem, hence it can only be a run-time process. However, If V and/or D and/or C is not known, then any processing is a compile-time process because we actually cannot solve the problem (some might say, what if C and V are given, then we assume that the domains of the variables contain all possible values, solve the problem at compile time, then at runtime we find a subset of the solution set which satisfy the actual domains of the variables. One can do that, but finding a subset of the solution set satisfying the domains of the variables is an NP-complete problem that need to be solved).

On the other hand, the generation of implied constraints cannot be done unless a subset of constraints C is known and obviously a subset of variables V. However, C may still contain non-domain variables, which are unknown or a when subset of D is not known then the generation of implied constraints is a compile-time process and the generation time is not an issue to be considered. However, if D, C, and D are also available, i.e., ground then the generation of implied constraints is a run-time process and the generation time is to be accounted for the overall solving time. Furthermore, if the CSP is supposed to be solved using some consistency algorithm plus a search algorithm (which might include consistency techniques also) then the run-time generation of implied constraints can be done prior to the start of the search (*pre-search generation*) as well during the search itself, (*within-search generation.*) One is also faced with the choice of whether to feed the generated implied constraints back to the original set of constraints, and use them to generate more implied constraint, or not to use them further .

3.2 Definitions and Terminology

Definition 3.4 *Mathematical Programming (MP):*
In MP*, one tries to find an extreme (i.e., maximium or minimium) point of a function $f(x_1, x_2, ..., x_n)$, which satisfies a set of constraints of the form $g(x_1, x_2, ..., x_n) \geq b$*

Definition 3.5 *Linear Programming (LP):*
LP *is a specialization of MP, where 1 f – to be called* objective function– *and the set of constraints are linear. The general formulation is as follows:*

$$\begin{array}{l} Objective\ function: \\ max/min\ c_1X_1 + c_2X_2 + ... + c_nX_n \\ Subject\ to: \\ a_{i1}X_1 + a_{i2}X_2 + ... + a_{in}X_n\{\leq, =, \geq\}b_i, i = 1, ..., n \end{array} \tag{14}$$

where all variables are either positive or negative or unrestricted.

Definition 3.6 *Integer Linear Programming (ILP):*
ILP *is a specialization of LP, where all the variables can only take integer values.*

Definition 3.7 *Mixed Integer Linear Programming (MILP):*
MILP *is a specialization of LP, where some but not all of the variables can only take integer values. Problems in this class are NP-complete.*

Definition 3.8 *Linear Relaxation (LR):*
LR *is the result of relaxing the integrality requirements in an ILP or MILP formulation. Near optimal solutions are found by solving the 0relaxed LP formulation.*

Definition 3.9 *Standard Form:*
an LP formulation is in standard form *if*

1. *all constraints are equality constraints*
2. *all variables have non negativity sign restriction*

Here are some definitions of different kinds of consistency ([39]):

Definition 3.10 *A problem is* (i, j)-consistent *[14] iff it has non-empty domains and any consistent instantiation of* i *variables can be extended to a consistent instantiation involving* j *additional variables.*

Definition 3.11 *A problem is* k-consistent *iff it is* $(k - 1, 1)$*-consistent.*

Definition 3.12 *A problem is* strongly k-consistent *iff it is* $(j, 1)$*-consistent, for all* $j < k$.

Definition 3.13 *A problem is* node-consistent *iff for all variables all values in its domain satisfy the constraints on that variable.*

Definition 3.14 *A problem is* arc-consistent *(AC) iff it is* $(1, 1)$*-consistent.*

Definition 3.15 *A problem is* path-consistent *(PC) iff it is* $(2, 1)$*-consistent.*

Definition 3.16 *A problem is* strong path-consistent *iff it is* $(j, 1)$*-consistent for all* $j \leq 2$.

Definition 3.17 *A problem is* path inverse consistent *(PIC) iff it id* $(1,2)$-*consistent.*

Definition 3.18 *A problem is* neighborhood inverse consistent *(NIC) iff any value for a variable can be extended to a consistent instantiation for its immediate neighborhood.*

Definition 3.19 *a (non-binary) CSP is* Generalized arc-consistency *(GAC) iff for any variable in a constraint and value that is assigned, there exists compatible value for all the other variables in the constraint.*

4 Related Work

Simplex Method

The simplex method is a well established method for solving linear programs in standard format. The algorithm starts with an initial basic feasible solution and tests its optimality. If some optimality condition is verified, then the algorithm terminates. Otherwise, it tries an adjacent basic feasible solution which has a better objective value, and optimality is tested again. The entire process is repeated till an optimal solution is found. However, since the simplex method works only for problems in standard form a transformation is required to map the other linear constraints to ones in a standard form. In fact, every LP can be transformed into one in standard form by the following transformation rules:

- $a_1X_1 + a_2X_2 + ... + a_nX_n \leq b$ can be transformed into an equality one by introducing a slack variable S, resulting in $a_1X_1 + a_2X_2 + ... + a_nX_n + S = b$
- $a_1X_1 + a_2X_2 + ... + a_nX_n \geq b$ can be transformed into an equality one by introducing an excess variable E, resulting in $a_1X_1 + a_2X_2 + ... + a_nX_n - E = b$
- a variable $X_i : X_i \leq 0$ is replaced by $X_iprime : X_i = -X_iprime$
- an unrestricted variable X can be substituted by Y and Z such that $X = Y - Z$ and $Y, Z \geq 0$

This transformation rules can be viewed as a way of generating a particular class of implied constraints.

Inference Methods

Consistency Methods

Consistency methods (e.g., [3, 8, 12, 13, 25]) can be viewed as methods that generate implied constraints. Consistency methods are defined in such a way that if a value in a domain or a compound label in a constraint does not satisfy some property, then they can be eliminated by adding the appropriate implied constraints. For instance, if we have the constraints $X \in \{1, ..., 10\} \wedge X \geq 5$ then by maintaining *node* consistency the domain of the variable will be reduced to $\{5, ..., 10\}$. Here is another example that shows the difference of maintaining different levels of consistency:

Example 4.1 Suppose we have the following set of constraints $X, Y, Z \in \{1, 2\} \wedge X \neq Y \wedge Y \neq Z \wedge X \neq Z$. Maintaining arc consistency will not lead to any change in the variable domains, since the problem is inherently arc-consistent. However, the same problem can be stated using a global constraint *alldifferent* for which there is a special algorithm to maintain the consistency, which is stronger than arc consistency. Hence, having $X, Y, Z \in \{1, 2\} \wedge alldifferent([X, Y, Z])$ will result in unsatisfiable problem.

Cutting Planes

A LR of an ILP can be solved by the Chvatal-Gomory method [1] through the addition of *cutting planes*, which are new inequalities inferred from the original set of constraints. The cutting plane cut of part of the search space but does not eliminate any solution to the original problem. They strengthen the relaxation in order to get better bound on the optimal value when the relaxation is solved.

Example 4.2 Suppose we have two variables X and Y each having domain $\{0, 1\}$ and our problem is to maximize $2X + 2Y \leq 1$ then some cutting planes are $X + Y \leq 1/3$ and $X + Y \leq 0$.

The Chvatal-Gomory procedure is a complete inference system, for integer linear constraints which generate cutting planes. The cutting planes are nothing but implied constraints from the original set of constraints.

Resolution Method

For propositional logical formulae in *conjunctive normal form* (CNF), the resolution rule of inference is a complete inference method. The rule is as follows:

$$\begin{array}{c} \psi \vee \phi_1 \\ \neg\psi \vee \phi_2 \\ \hline \phi_1 \vee \phi_2 \end{array} \tag{15}$$

Furthermore, for any constraint that includes only binary variables there exists an equivalent CNF formula. The values of the variables are interpreted as *True* and *False*. In [19], it is pointed out that for any constraint set in binary variables, a generalized version of resolution [20] can generate all valid inequality cuts.

NB-Resolution [15, 19]

Resolution can be extended to the logic of discrete variables. NB-clauses involve variables whose domains have arbitrary size and have the following form:

$$X_{i_1}/S_{i_1} \vee ... \vee X_{i_n}/S_{i_n} \tag{16}$$

The resolution rule of inference can be extended to NB-resolution for NB-clauses:

$$\frac{\begin{array}{c} X_i/S_1 \vee \phi_1 \\ X_i/S_2 \vee \phi_2 \\ . \\ . \\ . \\ X_i/S_k \vee \phi_k \end{array}}{\phi_1 \vee \phi_2 \vee ... \vee \phi_k} \tag{17}$$

where the premises are a minimal set of clauses such that $S_1 \cap ... \cap S_k = \emptyset$. In [19], it is shown that *k-resolution* achieves *k-consistency* and another restriction of resolution achieves *adaptive consistency* [9].

Constraint Handling Rules (CHR)

CHR is a committed-choice language consisting of guarded rules with multiple head atoms. Some of the rules define simplification of constraints. Other rules define propagation over constraints, which add implied constraints in a similar manner to the consistency methods.

Solvers Extensions

SoLeX [26] is a generic scheme, which consists of a set of symbolic rule-based transformations that extend constraint solvers so that they handle constraints involving new function symbols called *alien constraints*. SoLeX handles three classes of alien constraints:

1. introducing a name to an extensional definition of a function. For example, $3.x^2$ can be named $p(x)$ and hence $3.y^2$ will be named $p(y)$.
2. functions that are not handled by the solver such as *sin* for some of the arithmetic solvers.
3. functions with no defined meaning in the solver's domain.

SoLeX has four phases. First, the reduction phase adds semantic and syntactical information carried by the alien function. Second, implied valid constraints from the alien functions are added by the expansion phase. Third, the solving phase actually solves the constraints, after abstracting the remaining of the alien functions. Finally, the contraction phase undo the effect of abstraction and removes redundancies introduced by the expansion phase.

Solvers Collaborations

Collaboration of solvers can also be viewed as a way of generating implied constraints. For example, one can combine the ILP approach with constraint solvers of finite domain such as the work presented in [31]. These two solvers can exchange information, in terms of implied constraints, and collaborate to solve the problem. In [28], collaboration of solvers has been employed to solve non-linear polynomial constraints and a general scheme BALI [27] that allows for the integration, re-usability and collaboration of solvers in a domain independent way has been proposed.

Redundant Modeling

In [2], the authors present a way of collaboration between redundant models for the same problem, through what they called channeling constraints. The information exchanged through the channeling constraints is nothing but implied constraints, which help achieve more pruning very similar to the effect of adding redundant constraints.

5 Generation of Implied Constraints

5.1 Classification of Implied Constraints

We propose the following the classifications of implied constraints:

1. *Simplified implied constraint (or variable elimination)*: given a constraint C constraining n variables ($V = \{V_1, ..., V_n\}$), and a set S of other constraints among the n variables. We try to derive C_{imp} such that $\{C\} \cup S \rightarrow C_{imp}$, and C_{imp} constrains $n-1$ variables or less. Here is an example:

$$\begin{gathered} C \equiv X + Y + Z = 12 \\ V = \{X, Y, Z\} \\ S = \{Y > Z\} \\ C \cup S \rightarrow C_{imp} \equiv X + 2 * Y \geq 12 \end{gathered} \tag{18}$$

   We call this class C_1.

2. *Specialized implied constraint*: we try to derive a specialized implied constraint (global constraint), for which there exists an efficient implementation. Here is an example which derives the *alldifferent* constraint:

$$\begin{gathered} C = \{X \neq Y \wedge X \neq W \wedge X \neq Q \wedge X \neq Z \wedge Z \neq Y \wedge Q \neq Y\} \\ C \rightarrow alldifferent(X, Y, Z) \end{gathered} \tag{19}$$

   We call this class C_2.

3. *linear implied constraints inferred from linear and nonlinear constraints*: we try to imply a linear constraint provided a set of linear and nonlinear constraints. such as the case in the following example:

$$A^2 = 1\ mod\ 2 \rightarrow A = 1\ mod\ 2 \tag{20}$$

   We call this class C_3.

4. *Abstract implied constraint (Variable introduction)*: we try to abstract some relationship between some variables of the initial problem by introducing extra variables and try to infer implied constraints on the newly introduced variables. Here is an example:

$$\begin{gathered} V = \{X, Y, Z, W\} \\ C = \{X + Y \neq 2W - 3Z\} \\ C \rightarrow P = X + Y \wedge Q = 2W - 3Z \wedge P \neq Q \end{gathered} \tag{21}$$

   We call this class C_4.

5. *Abstraction of problems into different constraint systems:* given a problem formulation in a particular domain (integer, set, list, etc), we will try to derive some implied constraints on a different domain from the original domain. For instance, suppose X, Y, S *and* T are sets and in [17] one can state the constraints $S \cup T = X \wedge S \cap T = Y$. However, one can infer the constraints $\mid S \mid + \mid T \mid \geq \mid X \mid \wedge \mid Y \mid \geq 0$ which are in integer domain. We call this class C_5.

6. *Implied constraint over a new subset of variables*: given a set of constraints over a set of variables, we would like to infer a constraint over a subset of variable for which there exists no constraint among them.

$$\begin{gathered} V = \{X, Y, Z, W, M, N\} \\ C = \{X = Y + Z \wedge W = M - N \wedge Y + Z = 3.(M - N)\} \\ C \rightarrow X = 3W \end{gathered} \tag{22}$$

   We call this class C_6.

Assuming that we are able to generate such implied constraints, we should identify how to use them. We here present how to use each class.

1. Class C_1

5.2 The PRESS System

Overview

PRESS (PRolog Equation Solving System) [38] is a system which automatically solves symbolic equations in one or more variables. It is modular, with different kinds of equation-solving activity defined by *axioms* of the domain, and special-purpose problem-solving procedures called *methods*, and has in the past been tested on 'A' level examination questions, which it can mostly solve quite well, an overall 86.8 per cent success rate. The system generally tries to simplify the (set of) (in)equations it has been asked to solve until it has a solution. PRESS works in the domain of *R-elementary equations*, i.e., equations involving polynomials, and exponential, logarithmic, trigonometric, hyperbolic, inverse trigonometric and inverse hyperbolic functions over real numbers. PRESS has six major methods:

- *Isolation* is a method for solving equations involving only a single occurrence of an unknown. It achieves this by applying some isolation rewrite rules. Examples of such rewrite rules are:

$$\begin{array}{c} log_u V = W \to V = U^W \\ U - V = W \to U = V + W \end{array} \tag{23}$$

- *Polysolve* is a collection of polynomial methods thar are tried in turn depending on the characteristics of the polynomial such as being symmetric or ant-symmetric. For example, $x^4 - 4.x^2 + 3 = 0$ is identified to be quadratic in x^2 and the solution $x^2 = 1 or x^2 = 3$ is found.

- *Collection* is a method to reduce the number of occurrences of the unknown. Example of such rewrite rule is $(U + V).(U - V) \to U^2 - V^2$.

- *Attraction* is a method for bringing the occurrences of the unknown closer. For instance the rule $log_u V + log_u W \to log_u(V.W)$ is an attraction rule.

- *Homogenization* parses the equation and collects the terms which contain the unknown that are non algebraic in the unknown. Then it tries to replace such terms by some algebraic function of some single term. For instance, $(e^x)^3 - 4.e^x + 3/e^x = 0$ can be changed to $y^3 - 4.y + 3/y$ where y is substituted for e^x.

- *Function swapping* is a collection of methods that transform an equation according to preference. For instance, $\sqrt{x + y} = 2$ is transformed to $(x + y)^2 = 4$.

Since PRESS is modular, it seemed like a good candidate for our initial experiments in deriving implied constraints.

Improvements to the Press System

We modified PRESS is several ways in order to better suit our purposes:[1]

1. (Trivially) we updated the system to run on current versions of Prolog (sicstus version 3).

2. We extensively modified the system to allow it to solve inequalities. Rather than extending the existing inequality module, which is rather minimal, we chose to generalise the equation-solving procedures, for example finding roots of polynomials, to deal with inequalities. A key component of this solution was to extend the notion of substitution from the preexisting substitution using equalities — $x = B \vdash \phi \to \phi[B/x]$ — to inequalities. Care must be taken over the direction of the inequalities, and the sign of

[1]The following files were changed, from the distribution version: ./pressdir/toplevel/sim, ./pressdir/toplevel/simeq, ./pressdir/pressjunk/filin, ./pressdir/toplevel/solve, ./pressdir/axioms/ineqis.ax, ./pressdir/methods/poly, ./util/struct.pl.

any coefficients multiplying the substituted variable. The following table describes which substitutions using inequalities can be made, and how the principal connective is effected.

Expression	$L < R$	$L \leq R$	$L > R$	$L \geq R$	$L = R$
Substitution					
$A < B$	$\times$	$\times$	$>$	$>$	$>$
$A \leq B$	$\times$	$\times$	$>$	$\geq$	$\geq$
$A > B$	$<$	$\leq$	$\times$	$\times$	$<$
$A \geq B$	$<$	$\leq$	$\times$	$\times$	$\leq$
$A = B$	$<$	$\leq$	$>$	$\geq$	$=$

The table shows the main connective when an occurrence of A in the left hand side of the formula has been replaced with B. An $\times$ in a cell means that that substitution cannot be made. Substitution on the right hand side is defined in terms of substitution on the left hand side.

As when rewriting under implication and negation, each position in the expression has a *polarity.* The polarities of the subterms of B in $A - B$ is the opposite of the polarities of the subterms of B in $A + B$. Likewise for $A - 2B$ compared to $A + 2B$. Substitution of terms which are in a position of negative polarity reverses the inequality of the substituting term. A notion of unification on relations is defined to ensure that the principal connective in a substituting formula is compatible with the main connective in the expression on which the substitution is operating, and what the resulting connective should be.

We plan the following improvements in the very near future:

1. Provide PRESS with relevant theory for whatever constraint domain we are tackling.

2. PRESS tries to solve (in)equations. If it cannot find a solution, considers that it has failed, and backtracks to the most recent choice point. In fact, although the result of its manipulations may not constitute a "solution" to the problem, they may be very useful implied constraints. We will therefore modify the system with *acceptance criteria*, which are predicates applied at the leaves of its derivation tree[2] to determine if the derived formula should be kept as a candidate implied constraint. If it is accepted in this way, then it is recorded in Prolog's internal database and retrieved when PRESS has finished its attempt to find a solution.

3. PRESS is heavily biased to finding a *solution.* The guidance may be too tight and thereby miss some implied constraints. We may have to weaken this guidance.

[2]We can represent an attempted derivation as a tree. Each choice point creates a branch. The leaves are the formulae reached during deduction to which it could do no more and hence backtracked.

4. Allow PRESS, in tightly-controlled circumstances, to make external calls to a theorem prover (e.g. *Clam*, *Otter*) or a computer algebra system (e.g. *Maple*) in order to further simplify or prune some constraints.

Experiments with the Press system

The following table shows a collection of examples. Some of the examples have been encountred in [35, 32, 37]. The first column shows the class of the implied constraints, the second the set of constraints and the last is the implied constraints.

Class	*Constraints*	*Implied Constraints*
C_1	$A+B+C=12$ $\wedge A \leq B$ $\wedge B \leq C$	$C \geq 4 \wedge A \leq 4$
C_2	$X \neq Y \wedge Y \neq Z \wedge X \neq Z$	$alldifferent([X,Y,Z])$
C_3	$A^2+B^2+C^2=12$ $\wedge A \geq 0$ $\wedge A \leq B \wedge B \leq C$	$(C \geq 2 \vee C \leq -2) \wedge (A \leq 2 \vee A \geq -2)$
C_3	$X*Y*Z=8 \wedge Y \leq Z$ $\wedge Z \leq 2$	$X \geq 2$
C_4	$A+B \neq C+D$	$X=A+B \wedge Y=C+D \wedge X \neq Y$
C_5	$S \cup Y = X$	$\mid S \mid + \mid T \mid \geq \mid X \mid$
C_6	$X=N-M \wedge Y=P-N$ $\wedge Z=P-M$	$Z=X+Y$
C_6	$X=A+B \wedge Y=C+D$ $\wedge A+C=1 \wedge B+D=3$	$X+Y=4$
C_6	$X=A+B \wedge Y=C+D$ $\wedge A+B \neq C+D$	$X \neq Y$

Our task is to check how much of these implied constraints can PRESS produce. Here are the results:

Class	*Constraints*	*Pressresults*
C_1	$A+B+C=12 \wedge A \leq B \wedge B \leq C$	*yes*
C_2	$X \neq Y \wedge Y \neq Z \wedge X \neq Z$	*no*
C_3	$A^2+B^2+C^2=12 \wedge A \geq 0 \wedge A \leq B \wedge B \leq C$	*yes*
C_3	$X*Y*Z=8 \wedge Y \leq Z \wedge Z \leq 2$	*yes*
C_4	$A+B \neq C+D$	*no*
C_5	$S \cup Y = X$	*no*
C_6	$X=N-M \wedge Y=P-N \wedge Z=P-M$	*no*
C_6	$X=A+B \wedge Y=C+D \wedge A+C=1 \wedge B+D=3$	*no*
C_6	$X=A+B \wedge Y=C+D \wedge A+B \neq C+D$	*no*

In addition to being modular and easy to update the PRESS system has the following advantages:

- The PRESS system handles linear and nonlinear arithmetic constraints.
- The substitution mechanism of the PRESS system and its *isolation, collection and attraction* rules allow it to handle very complex nonlinear constraints in a very nice way.
- PRESS can generate some implied constraints in classes C_1 and C_3
- The PRESS system can be easily updated to handle constraints over other domains, such as set domain by adding the appropriate set axioms coded as rewrite rules.

However, the PRESS system suffers from the following problems:

- PRESS is heavily biased to finding a *solution.* The guidance may be too tight and thereby miss some implied constraints in the classes C_1 and C_3.
- PRESS can't handle classes C_2, C_4, C_5, and C_6.

5.3 the clp(q,r) solver

Overview

The clp(q,r) solver [22, 23] of Sicstus Prolog [21] solves linear equations over rational or real valued variables, employs a decision algorithm for linear inequalities which derives implied constraints, handles linear dis-equations, and uses the lazy treatment for nonlinear equations, i.e. it collects them hoping that with the addition of further linear constraints they might get simple enough to solve.

Experiments with the clp(q,r) solver

The clp(q,r) as the PRESS system try to actually solve the set of constraints provided, however what we were interested in is the implied constraints generated rather than the solutions. We carried out the same experiments as with the PRESS system and here are the results:

Class	$Constraints$	$Pressresults$
C_1	$A+B+C=12 \wedge A \leq B \wedge B \leq C$	no
C_2	$X \neq Y \wedge Y \neq Z \wedge X \neq Z$	no
C_3	$A^2+B^2+C^2=12 \wedge A \geq 0 \wedge A \leq B \wedge B \leq C$	no
C_3	$X*Y*Z=8 \wedge Y \leq Z \wedge Z \leq 2$	no
C_4	$A+B \neq C+D$	no
C_5	$S \cup Y = X$	no
C_6	$X=N-M \wedge Y=P-N \wedge Z=P-M$	yes
C_6	$X=A+B \wedge Y=C+D \wedge A+C=1 \wedge B+D=3$	yes
C_6	$X=A+B \wedge Y=C+D \wedge A+B \neq C+D$	yes

As it can be seen from the experiments, the clp(q,r) solver has a great potential to handle class C_6. However, it cannot handle classes C_1, C_2, C_3, C_4, and C_5.

5.4 The *Otter* system

Otter (I need refs) is an automated deduction system. Resolution and paramodulation are the basis of its inference rules. The theorem to be proven by Otter should be stated in first-order logic with equality.

Overview

Experiments with the Otter System

6 Evaluation of Implied Constraints

It is important to know when to add implied constraints and when not as such implied constraints do not always save the search effort. If the set of constraints are linear and the domains of variables are real, then there exist effective methods that can solve that, and any generation of implied constraints will not be useful. If the constraints are linear and nonlinear and the problem is supposed to be solved by MIP methods, then generating constraints of the classes C_3 and C_4 will be beneficial. If we have solvers over different domains cooperating to solve a hybrid CSP, then generating implied constraints of class C_5, may help. Our major focus, though, is on finite domain CSPs –where consistency techniques and search are used to solve them – for which all classes (C_1, C_2, C_3, C_4, C_5 and C_6) of implied constraints can be generated and may or may not be useful in the sense that adding implied constraints to the original problem may or may not improve the efficiency. In order to evaluate the implied constraints the domains of the variables should be provided in addition to the set of variables and the set of constraints. Thus, making the evaluation a runtime process, and hence effective methods should be developed. We will first show that evaluation of implied constraints is dependent on the instance data. The domains of the variables should be known in order to judge if an implied constraint will lead to more pruning or not. Since adding implied constraints to the original problem changes the topology of the constraint hyper-graph/(primal and dual) graph, we will study the change to the properties of the constraint hyper-graph and their relationship to local consistency and backtrack-free search. We will also try to understand the effect of adding implied constraints to the original problem and the level of local consistency, i.e., we will try to investigate the following question: Given a problem P_1 and adding implied constraints to P_1 will yield a new problem P_2. Now assume we maintain a certain level of consistency for problem P_2. What certain level of consistency is achieved for P_1? Is it a stronger level of consistency? Is it the same level of consistency? What are the conditions and how to predict them beforehand? The exploration of this question will be carried out for each class of implied constraints seperately.

6.1 Why is the Evaluation Instance Data Dependent?

Assume we have an original CSP involving the set of variables V, the set of constraints C and the set of domains D is not yet known. Suppose further that

$C \vdash R$, i.e., R is a redundant constraint. Assume further that we are trying to maintain a certain level of consistency by algorithm L. Now, if there exists a method M that takes as input V, C, R, and L and returns *yes* if the redundant constraint will lead to more pruning and *no* otherwise. We can always construct D in a way that contradicts the answer returned by M in the following manner; if the answer returned by M is yes then we simply choose the domains D in a way so that it is inherently consistent according to L and thus R won't lead to any further pruning. If the answer returned by M is no, then we simply choose the domains D in a way so that it is inherently inconsistent according to L and thus R will lead to some further pruning. Therefore, such an M does not exist and the domains of the variables should be used as a parameter if we wish to find such an M.

6.2 Graph-related Concepts

Hyper-graphs are a generalization of graphs. In a hyper-graph, each hyper-arc may connect more than two nodes. Every CSP, with the set of variables V and the set of constraints C, can be associated with a constraint hyper-graph where V will be the set of nodes and for every constraint c in C, there is an associated hyper-arc (drawn as a region) among the variables constrained by c. When the constraints are binary, the CSP will be represented by a graph. A *primal-constraint graph* represents variables by nodes and associates an arc with any two nodes involved in the same constraint. A *dual-constraint graph* represents the variables involved in a constraint by a node and associates a labeled arc with any two nodes who share some variables. The arcs are labeled with the shared variables. The dual graph transforms a non-binary CSP into a special type of binary CSP where the variables involved in a constraint in the non-binary CSP are represented by a variable in the binary CSP with their domains ranging over all combinations permitted by the corresponding constraint, and any two adjacent nodes in the dual graph, their shared variables should have the same value. In [6], the author surveys some of the methods that relate the level of local consistency and backtrack-free search based on the topological features of the primal constraint graph. For a binary CSP, if the constraint graph is a *tree* then it can be solved in a linear time ([11, 29, 7].)

6.3 Class C_1

...

6.4 Class C_2

...

6.5 Class C_3

...

6.6 Class C_4

Given an initial CSP involving the set of variables V and the set of constraints C, the implied constraints of class C_4 will introduce a new set of variables V_{new} and new set of constraints C_{new} to the original CSP. Thus our new problem CSP_{new} will be involving the set of variables $V \cup V_{new}$ and the set of constraints $C \cup C_{new}$. Solving CSP_{new} and projecting the answer on the set of variables V is in fact a solution to the CSP. The set of variables V_{new} is redundant in the sense that the removal of V_{new} together with all the constraints connecting them C_{new} does not change the set of solutions to the original problem CSP. In [33], V_{new} is called the set of *hidden* variables while V is the set of *visible* variables. Furthermore, in [33], sufficient conditions for hidden variables are explored. We will try to use the results found there ...

6.7 Class C_5

...

6.8 Class C_6

...

7 Work Plan

The following is a proposed timetable of the tasks that need to be done for next year.

- Carry out the same experiments with the otter system. Should be done by September the 15th, 2000 .
- carry out a deep study about the evaluation of implied constraints in general. Should be done by October 30th, 2000.
- Study the evaluation of two class of implied constraints (to be chosen later). Should be done by April 2001.
- Propose a hybrid system (which may have ATP components as well as other 0 components) that has the capabilities of generating implied constraints of the chosen two classes. Should be done by June 15th, 2001.
- Implement a prototype system. Should be done by August, 30th, 2001. .
- Selecting the fast techniques to be used if the generation is at run-time. Should be done by October the 15th, 2001.

8 Conclusion

In this report, we tried to define the issues related to the automatic generation and evaluation of implied constraints. After defining compile-time versus run-time generation of implied constraints, we presented a classification of implied constraints and experimented with some of ATP technologies to see how much of these classes can they handle. We then defined the problem of evaluating implied and pointed out to some possible future directions.

References

[1] E. Balas, S. Ceria, G. Cornuejols, and N.R. Natraj. Gomory cuts revisited.*Operations Research Letters*, 19, 1996.

[2] B.M.W. Cheng, K.M.F. Choi, J.H.M. Lee, and J.C.K. Wu. Increasing constraint propagation by redundant modeling: an expertise report. xxx.

[3] M.C. Cooper. An optimal k-consistency algorithm. *Artificial Intelligence,* 41:89-95, 1989.

[4] M. Carlson, G. Ottosson and B. Carlson. An open-ended finite domain constraint solver. *Proc. Programming Languages: Implementations, Logics, and Programs,* 1997.

[5] Ph. Codognet and D. Diaz. Compiling constraints in clp(FD). *J. of Logic Programming* 27(3):185–226, 1996.

[6] R. Dechter. Constraint Networks. *In Encyclopedia of Artificial Intelligence*, 2nd edition, 1992, John Wiley & Sons, Inc., pp. 276-285.

[7] R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, Vol. 38, No. 1, pp. 1-38, 1987.

[8] R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence,* 34:1-38, 1988.

[9] R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. In Kanal and Kumar, editors, *Search in Artificial Intelligence.* Springer-Verlag, 1988.

[10] G.B. Dantzig. *Linear programming and extensions.* Princeton University Press, Princeton, N.J., 1963.

[11] E.C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, Vol. 92, No. 1, pp., 24-32, 1982.

[12] E.C. Freuder. Synthesizing constraint expressions. *Comm. ACM,* 21:958-966, 1978.

[13] E.C. Freuder. A sufficient condition for backtrack-bounded search. *J. ACM,* 32:755-761, 1985.

[14] E. C. Freuder. Backtrack-free and backtrck-bounded search. In Kanal and Kumar, editors, *Search in Artificial Intelligence*. Springer-Verlag, 1988.

[15] A.M. Frisch. Solving constraint satisfaction problems with NB-resolution. In S. Muggleton, D. Michie and Luc De Raedt, editors, *Machine Intelligence 16*, 2000. Electronic Transactions in Artificial Intelligence.

[16] R.S. Garnfield and D.S. Nemhauser. *Integer programming.* Jhon Wiley and Sons, New York, 1972.

[17] C. Gervet. Interval propagation to reason about sets: Definition and implementation of a practical language. *Constraints* 1(3):191-244, 1997.

[18] L. Getoor, G. Ottosson, M. Fromherz and B. Carlson. Effective redundant constraints for online scheduling. In *Proceedings of AAAI'97,* 1997.

[19] J.N. Hooker. Constraint satisfaction methods for generating valid cuts. xxx.

[20] J.N. Hooker. Generalized resolution for 0-1 inequalities. *Annals of Mathematics and Artificial Intelligence*, 6, 271-286, 1992. xxx.

[21] C. Holzbaur. OFAI clp(q,r) Manual, Edition 1.3.3. Austrian Research Institute for Artificial Intelligence, Vienna, TR-95-09, 1995.

[22] C. Holzbaur. A High-Level approach to the realization of CLP languages, *in Proceedings of the JICSLP92 Post-Conference Workshop on Constraint Logic Programming Systems*, Washington D.C., 1992.

[23] C. Holzbaur. A Specialized, incremental solved form algorithm for systems of linear inequalities, *Austrian Research Institute for Artificial Intelligence*, Vienna, TR-94-07, 1994.

[24] J. Jaffar, M. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503-582, May/July 1994.

[25] U. Montari. Networks of constraints: Fundamental properties and applications to picture processing. *inform. Sci.,* 7:95-132, 1974.

[26] E. Monfroy and C. Ringeissen. An open automated framework for constraint solver extension: the SoLeX approach. *Fundamenta Informaticae* 34, 1-20, IOS Press, 1999.

[27] E. Monfroy. The constraint solver collaboration language of BALI. xxx.

[28] E. Monfroy, M. Rusinowitch, and R. Schott. Implementing non-linear constraints with cooperative solvers. xxx.

[29] A.K. Mackworth and E.C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, Vol. 25, No. 1, 1984.

[30] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99-118, 1977.

[31] G. Ottosson. Integration of constraint programming and integer programming for combinatorial optimization. *Uppsala Theses in Computing Science 33.* 143 pp. Uppsala. ISSN 0283-359X, ISBN 91-506-1396-0.

[32] L. Proll and B.M. Smith. ILP and constraint programming approaches to a template design problem. University of Leeds, TR-97.16

[33] F. Rossi. Redundant hidden variables in finite domain constraint problems, in *Constraint Processing*, M. Meyer ed., Springer-Verlag, LNCS 923, 1995.

[34] F. Rossi. Existential variables and local consistency in finite domain constraint problems, *in Proc. CP96*, Springer Verlag, LNCS 1118, 1996.

[35] B.M. Smith, K. Stergiou, and T. Walsh. Modelling the Golomb ruler problem. In *Proceedings of the IJCAI99 Workshop on Non-Binary Constraints.* International Joint Conference on Artificial Intelligence, 1999.

[36] G. Smolka. The Oz programming model. In Jan Van Leeuwen, editor, *Computer Science Today* LNCS, No. 1000, Springer Verlag, 1995.

[37] K. Stergiou and T. Walsh. The difference all-different makes. xxxx

[38] L. Sterling, A. Bundy, L. Byrd, R. O'keefe, and B. Silver. Solving symbolic equations with PRESS. *J. Symbolic Computation*, 7, 71-84, 1989.

[39] E.P.K. Tsang. *Foundation of constraint satisfaction*. Academic Press, 1993.

[40] P. Van Hentenryck. *The* OPL *Optimization programming language.* The MIT Press, 1999.