Knowledge Reuse for Open Constraint-Based Inference

Le Chang
Department of Computer Science
University of British Columbia
2366 Main Mall, Vancouver, BC, Canada
lechang@cs.ubc.ca

Alan K. Mackworth
Department of Computer Science
University of British Columbia
2366 Main Mall, Vancouver, BC, Canada
mack@cs.ubc.ca

ABSTRACT

Open constraint programming, including open constraint satisfaction (Open COP) and open constraint optimization (Open COP), is an extended constraint programming framework designed to model and solve practical problems with open-world settings. We extend open constraint programming to the Open Constraint-Based Inference (Open CBI) framework based on the unified semiring-based CBI framework. The Open CBI framework subsumes both Open CSP and Open COP and also provides extensibility to cover more application domains. Furthermore, the Open CBI framework relaxes the assumption of domain value being incrementally discovered and revealed in non-decreasing order of cost, as required in open constraint programming. We have shown that junction tree representations and junction tree algorithms can be applied to handle Open CBI problems. We show in this paper that the junction tree representation is a suitable graphical model to reuse the intermediate computational results to subproblems. We also proposed consistency maintenance algorithms for junction tree to Open CBI problems with domain value addition and removal. We analyze and show that both answering the satisfiability or the optimal weight of the problem and finding total assignment of variables can be achieved in time that is linear in the size of the junction tree, which is fractionally smaller than the time needed to enforce the junction tree consistency from scratch. We also discuss directions of future research in applying graphical models to problems with open-world settings.

General Terms

Algorithms, Theory

Keywords

Open Constraint Satisfaction, Open Constraint Optimization, Junction Tree, Inference Algorithms, Constraint-Based Inference, Knowledge Reuse

1. INTRODUCTION

Constraint programming [25] is a general and powerful paradigm for modeling and solving combinatorial problems from various research fields from theoretical to applied. These combinatorial problems cover a wide range of disciplines in artificial intelligence, computer science, and operations research. The success of the constraint programming framework, with advances in constraint satisfaction, soft constraints, and constraint logic, has already been proven in numerous theoretical and practical problems such as combinatorial optimization, machine vision, planning, scheduling, fault diagnosis, configuration, and system simulation.

However, there still remain many real world problems that are difficult to represent and solve in the basic constraint programming frameworks because of the closed-world assumption, i.e. all variables, domains, and constraints are required to be completely known and fixed from the beginning of problem modelling and solving. This assumption does not always hold in real application scenarios: a user posts new constraints to or removes existing constraints from the problem in the middle of problem solving; new options that are modelled as domain values of a variable in constraint programming become available or unavailable because of some external events; it is neither feasible nor efficient to retrieve all of a huge number of possible options from different remote sites, e.g. flight information residing in different database servers of different airline companies.

The limitation of the closed-world assumption for constraint programming frameworks has drawn more and more attentions from research communities. Open constraint programming [12], including open constraint satisfaction (Open CSP) and open constraint optimization (Open COP), is an extended framework to model and solve practical applications with open-world settings. Many of these applications are inspired by the increasing use of the Internet, for example, locating additional suppliers for supply chain management, looking for additional bidders for online bidding applications, or planning trips according to online information search engines.

Most approaches to open constraint programming are based on backtrack search. Unsolvable failures in Open CSP and over-threshold weights in Open COP are used to trigger the acquisition of new values from variables with open domains. Two sound and complete search algorithms, *o-search* and *fo-search* were proposed in [12] for Open CSPs. By assuming that domain values of a variable in Open COP are always revealed in a decreasing order of preference, two backtrack search based algorithms for possibilistic and weighted Open COP were proposed as well in [12]. They are proven to be sound and complete and produce an optimal solution with the minimal number of queries. Recently, dynamical programming based approaches, such as DPOP [23] and ODPOP [24], have been applied to solve open constraint programming problems with good success.

As a fundamental approach in classic constraint programming, backtrack search has been already shown in many application domains that has advantages for solving large scale problems with limited space resources. Inference approaches, including arc consistency [19] and other local consistency algorithms, also play an important role in classic constraint programming. They are used individually or integrated with search algorithms to accelerate the search and improve the flexibility of answering multiple queries by reusing the computational results. We have shown that constraint satisfaction and probabilistic inference can be seen as special cases of the constraint-based inference (CBI) framework [4]. Junction tree algorithms in probabilistic inference [27, 18, 13], constraint satisfaction [11, 28], and information theory [1] can all be generalized with the CBI framework as an exact inference algorithm. Junction tree algorithms can be seen as memorized dynamic programming [8], where solutions for subproblems are memorized for later use. A junction tree is a structure that efficiently divides the original problem into subproblems. These properties of junction tree algorithms make it suitable in open constraint programming, especially when the solver cannot receive new domain values from variables in its preferred order or variables cannot reveal their top choices because of limited resource or privacy concerns. We show in this paper that the junction tree is an ideal graphical model to re-organize open constraint programming problems that facilitates the knowledge reuse in solving these problems.

2. BACKGROUND

2.1 Constraint-Based Inference

Constraint-Based Inference (CBI) is an umbrella term for various superficially different problems. It concerns the automatic discovery of new constraints from a set of given constraints over individual entities. New constraints reveal undiscovered properties about a set of entities. A constraint here is seen as a function that maps possible value assignments to a specific value domain. Many practical problems from different fields can be seen as constraint-based inference problems. These problems cover a wide range of topics in computer science research, including probabilistic inferences, decision-making under uncertainty, constraint satisfaction problems (CSP), propositional satisfiability problems (SAT), decoding problems, and possibility inferences.

A CBI problem is defined in terms of a set of variables with values in finite domains and a set of constraints on these variables. We use commutative semirings to unify the representation of constraint-based inference problems from various disciplines into a single formal framework [4], based on the synthesis of the existing generalized representation frameworks [3, 26, 16] and algorithmic frameworks [10, 14, 1] from different fields. Formally:

DEFINITION 1. *(Constraint-Based Inference Problem) A constraint-based inference (CBI) problem* $\mathbf{P}$ *is a tuple* $(\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$ *where:*

- $\mathbf{X} = \{X_1, \cdots, X_n\}$ *is a set of variables*
- $\mathbf{D} = \{\mathbf{D_1}, \cdots, \mathbf{D_n}\}$ *is a collection of finite domains, one for each variable*
- $\mathbf{S} = \langle \mathbf{A}, \oplus, \otimes \rangle$ *is a commutative semiring*
- $\mathbf{F} = \{f_1, \cdots, f_r\}$ *is a set of constraints. Each constraint is a function that maps value assignments of a subset of variables to values in* $\mathbf{A}$

Inference in a CBI problem corresponds to computing a new constraint over a subset of variables given existing constraints.

The definition of the CBI framework is based on the commutative semiring concept. A commutative semiring $\mathbf{S} = \langle \mathbf{A}, \oplus, \otimes \rangle$ consists a set $\mathbf{A}$ and two binary operations, addition $\oplus$ and multiplication $\otimes$, which apply to the set $\mathbf{A}$. Both addition and multiplication operations have associative and commutative properties and have identity elements in $\mathbf{A}$. Most important, the multiplication operation has the distributivity property over the addition operation in a commutative semiring.

More specifically, we use $Scope(f)$ to denote the subset of variables that is in the scope of the constraint f. We use $\mathbf{D}_X$ to denote the value domain of a variable X. In the following sections, we use bold letters to denote sets of elements and regular letters to denote individual elements. Given a variable $X \in Scope(f)$, $Scope(f)_{-X}$ denotes the variable subset $Scope(f) \setminus \{X\}$. Given a value assignment $\mathbf{x}$ of variable subset $\mathbf{X}$ and $\mathbf{Y} \subseteq \mathbf{X}$, $\mathbf{x}_{\downarrow \mathbf{Y}}$ denotes the value assignment projection of $\mathbf{x}$ onto the variable subset $\mathbf{Y}$. Then we define the two basic constraint operators as follows.

DEFINITION 2. *(The Combination of Two Constraints) The combination of two constraints* f_1 *and* f_2 *is a new constraint* $g = f_1 \otimes f_2$*, where* $Scope(g) = Scope(f_1) \cup Scope(f_2)$ *and* $g(\mathbf{w}) = f_1(\mathbf{w}_{\downarrow Scope(f_1)}) \otimes f_2(\mathbf{w}_{\downarrow Scope(f_2)})$ *for every value assignment* $\mathbf{w}$ *of variables in* $Scope(g)$.

DEFINITION 3. *(The Marginalization of a Constraint) The marginalization of X from a constraint f, where $X \in Scope(f)$, is a new constraint $g = \bigoplus_X f$, where $Scope(g) = Scope(f)_{-X}$ and $g(\mathbf{w}) = \bigoplus_{x_i \in \mathbf{D}_X} f(x_i, \mathbf{w})$ for every value assignment $\mathbf{w}$ of variables in $Scope(g)$.*

According to the definitions of the CBI problem and the basic constraint operators, we define the abstract inference and allocation tasks for a CBI problem.

DEFINITION 4. *(The Inference Task for a CBI Problem) Given a subset of variables $\mathbf{Z} = \{Z_1, \cdots, Z_t\} \subseteq \mathbf{X}$, let $\mathbf{Y} = \mathbf{X} \setminus \mathbf{Z}$, the inference task for a CBI problem $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$ is defined as computing:*

$$g_{CBI}(\mathbf{Z}) = \bigoplus_{\mathbf{Y}} \bigotimes_{f \in \mathbf{F}} f \tag{1}$$

Given a CBI problem $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$, if $\oplus$ is idempotent, we can define the allocation task for a CBI problem.

DEFINITION 5. *(The Allocation Task for a CBI Problem) Given a subset of variables $\mathbf{Z} = \{Z_1, \cdots, Z_t\} \subseteq \mathbf{X}$, let $\mathbf{Y} = \mathbf{X} \setminus \mathbf{Z}$, the allocation task for a CBI problem $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$ is to find a value assignment for the marginalized variables $\mathbf{Y}$, which leads to the result of the corresponding inference task $g_{CBI}(\mathbf{Z})$. Formally, we compute:*

$$\mathbf{y} = \arg \bigoplus_{\mathbf{Y}} \bigotimes_{f \in \mathbf{F}} f \tag{2}$$

where $\arg$ is a prefix of operator $\oplus$. In other words, $\arg \oplus$ is an operator that returns arguments of the $\oplus$ operator. For example, when $\oplus = max$, $\arg \oplus = \arg max$ that returns a value assignment that leads to the maximal possible element in $\mathbf{S}$.

In general, $\otimes$ is a combination operator in CBI problems that combines a set of constraints into a constraint with a larger scope; $\oplus_{\mathbf{Y}} = \oplus_{\mathbf{X} \setminus \mathbf{Z}}$ is a marginalization operator that projects a constraint over the scope $\mathbf{X}$ into its subset $\mathbf{Z}$, through enumerating all possible value assignments of $\mathbf{Y} = \mathbf{X} \setminus \mathbf{Z}$.

Many CBI problems from different disciplines can be embedded into our semiring-based unifying framework [4]. These problems include the decision task and allocation task of CSP and SAT, Max SAT and Max CSP, Fuzzy CSP, Weighted CSP, probability assessment, most probable explanation (MPE), dynamic Bayesian networks (DBN), possibility inference with various t-norms, and maximum likelihood decoding. More specifically, we use the commutative semirings $\mathbf{S_{CSP}} = \langle \{\mathit{FALSE}, \mathit{TRUE}\}, \vee, \wedge \rangle$, $\mathbf{S_{WCSP}} = \langle \mathbb{R}^+ \cup \{0\}, \min, + \rangle$, $\mathbf{S_{PoCSP}} = \langle [0,1], \min, \max \rangle$, and $\mathbf{S_{PrCSP}} = \langle [0,1], \max, \times \rangle$ to represent classic CSP, Weighted CSP, Possibility CSP, and Probabilistic CSP, respectively, in the CBI framework.

2.2 Junction Tree for CBI

In many cases, we can use graphical models to represent CBI problems. A widely studied and used graphical representation is primal graph. The primal graph representation of a CBI problem is an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{V_1, \cdots, V_n\}$ is a set of vertices, each vertex V_i corresponding to a variable X_i of the CBI problem[1]; and $\mathcal{E} = \{(V_i, V_j) | V_i, V_j \in \mathcal{V}\}$ is a set of edges between V_i and V_j. There exists an edge (V_i, V_j) if and only if corresponding variables X_i and X_j appear in the scope of the same constraint. A moralized graph of the Bayesian Network (BN), which is obtained by adding edges among vertices with the common child vertex in the corresponding BN, is an example of the primal graph representation of probabilistic inference problems. A constraint graph of a binary CSP is another example of the primal graph representation.

EXAMPLE 1. *Consider a constraint-based inference problem with 5 variables $V_1, \cdots, V_5$, $V_i \in \{0, 1\}$. There are 3 constraints defined over these variables: $f_1(V_1, V_2, V_3)$, $f_2(V_2, V_3, V_4)$ and $f_3(V_3, V_5)$, which specify the set of tuples permitted by these constraints, respectively. An inference task in this example is to discover tuples permitted by the derived constraint over V_2 and V_3.*

Given the CBI problem described in Example 1, the corresponding primal graph is shown in Figure 1(a).

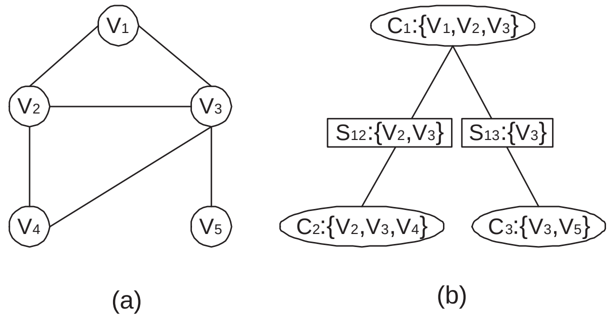

Figure 1: Graphical Representations of an CBI Problem described in Example 1. (a) Primal graph (b) Junction tree

Sometimes the primal graph of a CBI problem is re-organized as a secondary structure to achieve better computational efficiency. The junction tree is a widely used secondary structure in graphical models, especially in probabilistic reasoning. A junction tree is an undirected graph $\mathcal{T} = (\mathcal{C}, \mathcal{S})$. $\mathcal{C} = \{C_1, \cdots, C_n\}$ is a set of clusters, where each cluster C_i corresponds to an aggregation of a subset of vertices $V_{C_i} \subseteq \mathcal{V}$ in the primal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. $\mathcal{S} = \{S_{ij}, \cdots, S_{lm}\}$ is a set of separators between clusters, where S_{ij} is the separator of clusters C_i and C_j, corresponding to the vertices of

[1]In this paper, we sometimes use the same letter to represent a variable and its corresponding vertex if not specified.

$V_{C_i} \cap V_{C_j}$. In addition, the following junction tree properties have to be satisfied:

1. *Singly connected property*: $\mathcal{T} = (\mathcal{C}, \mathcal{S})$ is a tree;
2. *Running intersection property*: $\forall C_i, C_j \in \mathcal{C}$, $V_{C_i} \cap V_{C_j} \subseteq V_{C_k}$ holds for any cluster C_k on the path between C_i and C_j;
3. *Constraint allocation property*: For any constraint f of the CBI problem, $\exists C_i \in \mathcal{C}$ s.t. $Scope(f) \subseteq C_i$.

Given the CBI problem described in Example 1, the corresponding junction tree representation is shown in Figure 1(b).

Typically a junction tree is undirected. In some computational schemes, we pick one cluster as the root of the tree and assign directions to all separators. A separator $S_{ij} = (C_i, C_j)$ has a direction from C_i to C_j if C_i is in the path from the root to C_j. For each cluster C_i, $Parent(C_i)$ denotes the cluster that points to C_i; $Child(C_i)$ denotes the set of clusters that C_i points to. Given an arbitrary variable X_i and its corresponding vertex V_i, Theorem 1 shows that the subgraph of $\mathcal{T} = (\mathcal{C}, \mathcal{S})$ induced by X_i (or V_i) is a tree. This property is important to use junction tree in maintaining knowledge of open constraint programming problems. For any cluster $C_i \in \mathcal{C}$ of a junction tree $\mathcal{T} = (\mathcal{C}, \mathcal{S})$, the subgraph including C_i and all clusters that is lower than C_i and separators between them consist a tree as well. We call it **subtree rooted at** C_i.

THEOREM 1. *Given a junction tree $\mathcal{T} = (\mathcal{C}, \mathcal{S})$, constructed from a primal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a vertex V. Let $\mathcal{T}_\mathcal{V} = (\mathcal{C}_\mathcal{V}, \mathcal{S}_\mathcal{V})$, where $\mathcal{C}_\mathcal{V}$ is a subset of clusters that contain V and $\mathcal{S}_\mathcal{V}$ is a subset of separators that connect any two clusters in $\mathcal{C}_\mathcal{V}$. Then $\mathcal{T}_\mathcal{V}$ is also a tree. We call $\mathcal{T}_\mathcal{V}$ the* ***subtree induced by*** *V.*

PROOF. Given $\mathcal{T} = (\mathcal{C}, \mathcal{S})$ is a tree, any connected subgraph of $\mathcal{T}$ is also a tree. We then prove that $\mathcal{T}_\mathcal{V}$ is connected by contradiction. Assume that C_i and C_j are two clusters that contain V and not connected, then there must be a cluster C_k in the path between C_i and C_j and does not contain V, otherwise C_i and C_j are connected. The existence of C_k in the path between C_i and C_j is a contradiction to the running intersection property of junction tree. $\square$

The **width** of a junction tree is $max_{c \in \mathcal{C}} |X_c| - 1$. The **treewidth** of the primal graph $\mathcal{G}$ for a CBI problem, denoted by $w^*(\mathcal{G})$, is the minimum width over all possible junction tree representations. Treewidth is a key parameter in junction tree based algorithms because both the time and space complexities of junction tree algorithms are polynomial in the size of the junction tree, with a constant factor exponential in the treewidth. The junction tree is usually constructed from the primal graph representation through triangulating the primal graph. It is known that finding the optimal junction tree, in other words, finding minimal treewidth, is $\mathcal{NP}$-hard [2] in general. Several heuristic triangulation algorithms are discussed in [4] to construct sub-optimal junction tree in linear time. Details of heuristic triangulation algorithms and their empirical evaluations can be found in [15].

3. OPEN CONSTRAINT PROGRAMMING AS CONSTRAINT BASED INFERENCE

We extend the definition of both Open CSP and Open COP [12] and generalize them into the Constraint-Based Inference (CBI) framework using the semiring concept as follows:

DEFINITION 6. *(Open Constraint-Based Inference) An open constraint-based inference (Open CBI) problem is a possibly unbounded and ordered set $\{CBI(0), CBI(1), \cdots\}$ of constraint-based inference problems, where $CBI(i)$ is defined by a tuple $(\mathbf{X}, \mathbf{D}(i), \mathbf{S}, \mathbf{F}(i))$ where:*

- $\mathbf{X} = \{X_1, \cdots, X_n\}$ *is a set of n variables*
- $\mathbf{D}(i) = \{\mathbf{D_1}(i), \cdots, \mathbf{D_n}(i)\}$ *is a set of discrete domains for $CBI(i)$*
- $\mathbf{S} = \langle \mathbf{A}, \oplus, \otimes \rangle$ *is a commutative semiring*
- $\mathbf{F}(i) = \{f_1(i), \cdots, f_r(i)\}$ *is a set of constraints. Each constraint $f_k(i)$ is a function that maps* ***current*** *domain value combination of variables in its scope to values in* $\mathbf{A}$

We assume that for any $CBI(i)$ and $CBI(i+1)$, there exists a $k \in \{1, \cdots, n\}$, such that either $\mathbf{D_k}(i) \subset \mathbf{D_k}(i+1)$ or $\mathbf{D_k}(i+1) \subset \mathbf{D_k}(i)$, and for all $l \in \{1, \cdots, n\}$ and $l \neq k$, $\mathbf{D_l}(i) = \mathbf{D_l}(i+1)$. In other words, we assume each time only one variable reveals to or retracts from the solver some of its domain values. All the domains of the other variables remain the same.

The solution to an Open CBI problem is such that for each $CBI(i)$ in the sequence, compute:

$$\mathbf{x}(i) = \arg \bigoplus_{\mathbf{X}} \bigotimes_{f \in \mathbf{F}(i)} f \quad (3)$$

The Open CBI framework is different from the open constraint programming framework in the following aspects: (1) it integrates Open COP, Open CSP, and probability reasoning problems, such as MPE (most probable explanation) that maximizes the joint probability distribution, into a unified Open CBI framework; (2) domain values are not only incrementally discovered in Open CBI, variables can remove already discovered domain values from the problem; (3) the variable owners (not necessarily the same as the solver) decide to report domain value changes; (4) variables do not always reveal domain values in non-decreasing order of cost, the computation goal is now to report the satisfiable or optimal assignment according to the solver's representation of the world at the current time.

THEOREM 2. *If domain values are incrementally discovered, i.e. no value is removed from its domain once revealed and $\mathbf{S} = \langle\{FALSE, TRUE\}, \vee, \wedge\rangle$, then the solution $\mathbf{x}(i)$to $CBI(i)$ is also the solution to $CBI(j)$, for all $j > i$.*

PROOF. $\mathbf{S} = \langle\{FALSE, TRUE\}, \vee, \wedge\rangle$, then each $CBI(i)$ in the sequence is a standard CSP. A solution to $CBI(i)$ is then a satisfiable assignment to the corresponding $CSP(i)$. Because domain values are incrementally discovered, allowed tuples for each constraint are incrementally discovered as well. For arbitrary $j > i$, each constraint in $CSP(j)$ contains the allowed tuples appear in the corresponding constraint in $CSP(i)$. The projection of the solution in $CSP(i)$ to each constraint is an allowed tuple in $CSP(i)$, so it is also allowed in $CSP(j)$. It implies that a solution $\mathbf{x}(i)$to $CBI(i)$ is also a solution to $CBI(j)$, for all $j > i$. $\square$

Theorem 2 indicates that if domain values are incrementally discovered, then the Open CSP is a special case of the Open CBI.

THEOREM 3. *Open Constraint Optimization Problem (Open COP) is a special case of Open CBI if supplied with appropriate semirings and composition of functions.*

PROOF. Given Definition 2 in [12], a Open COP is a tuple $(\mathbf{X}, \mathbf{C}, \mathbf{D}(i), \mathbf{R}, \mathbf{W}(i))$, where $\mathbf{X}$ and $\mathbf{D}(i)$ are variables and their domain at a specific time, which is the same as in the definition of Open CBI.

The difference between Open COP and Open CBI is then in the description of crisp and soft constraints. In Open COP, $\mathbf{C}$ is a set of constraints and $\mathbf{R}$ is a set of relations that correspond to constraints. These two items can be unified as a set of functions $\mathbf{F_b}$. Each function in $\mathbf{F_b}$ corresponds to a constraint in $\mathbf{C}$ and maps tuples in the corresponding relation to the minimal cost and other tuples to the maximal cost. $\mathbf{W}(i)$ can be seen implicitly as a set of unary functions $\mathbf{F_u}(i)$, where each function in $\mathbf{F_u}(i)$ corresponds to a variable in $\mathbf{X}$ (as its scope) and maps current domain values of this variable to weights described in $\mathbf{W}(i)$. If we restrict the function set $\mathbf{F}(i)$ of a Open CBI problem to be $\mathbf{F_b} \cup \mathbf{F_u}(i)$ and provide explicitly a semiring $\mathbf{S}$ associated with this problem, the definition of Open COP is identical to the definition of Open CBI. $\square$

COROLLARY 1. *Open Possibility COP is a special case of Open CBI, if semiring $\mathbf{S} = \langle[0, 1], \min, \max\rangle$ and the function set $\mathbf{F}(i)$ consists of two parts: (1) a subset of non-unary functions that map consistent tuples to 0 and inconsistent tuples to 1, and (2) a subset of unary functions that map domain values of variables to corresponding costs.*

COROLLARY 2. *Open Weighted COP is a special case of Open CBI, if semiring $\mathbf{S} = \langle[0, \infty), \min, +\rangle$ and the function set $\mathbf{F}(i)$ consists of two parts: (1) a subset of non-unary functions that map consistent tuples to 0 and inconsistent tuples to ∞, and (2) a subset of unary functions that map domain values of variables to corresponding costs.*

Theorem 2 and Theorem 3 claim that original open constraint programming [12] is a proper subset of the proposed Open CBI framework. If the assumptions of open constraint programming hold in the Open CBI settings, the existing approaches for open constraint programming work as well for Open CBI problems.

4. JUNCTION TREE FOR OPEN CBI

The major challenge raised in the Open CBI framework is to compute in a timely fashion the satisfiable or optimal solution under the change of domain values and constraint tuples. How to efficiently organize, reuse and maintain the knowledge is the key issue facing this challenge.

In the Open CBI framework, domain values of variables are revealed to and removed from the problem solver. Meanwhile, constraint tuples that includes the changing values are added to and removed from constraints with that variable in their scopes. However, the variables and constraints are assumed to be completely known beforehand and will not change during the problem solving for a Open CBI problem. In other words, we assume that in the Open CBI framework, the structure of the problem remain unchanged. It provides possibilities that we can re-organize the structure of the problem to cope with challenges raised in the open-world settings.

Junction tree is known as a secondary structure graphical model used to maintain intermediate computational results for subproblems and to answer multiple queries in probabilistic reasoning with the closed-world assumption. Given that probabilistic reasoning can be seen as a special case of constraint-based inference, we introduce here Junction Tree (JT) as a graphical model to minimize the computational efforts of solving individual CBI problems in the sequence of a Open CBI problem.

4.1 Construction of Junction Tree

Given an Open CBI problem $\mathcal{P}$ defined as $\{CBI(0), CBI(1), \cdots\}$ with $CBI(i) = (\mathbf{X}, \mathbf{D}(i), \mathbf{S}, \mathbf{F}(i))$, construction of a junction tree for $\mathcal{P}$ consists of three steps: (1) construction of a primal graph; (2) triangulation of the primal graph; and (3) identifying a junction tree from the triangulated graph.

4.1.1 Construction of Primal Graph

It is straightforward to construct a primal graph given the variable set $\mathbf{X}$ and the constraint set $\mathbf{F}$. First we enumerate all variables in $\mathbf{X}$. For each $X_i \in \mathbf{X}$, add a vertex V_i to the vertex set $\mathcal{V}$. Second we enumerate all constraints in $\mathbf{F}$. For each $f_k \in \mathbf{F}$ and $\forall X_i, X_j \in Scope(f_k)$, add a pair (V_i, V_j)

```
Input: A connected primal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, an ordering
    $\sigma = (V_1, \cdots, V_n)$
Output: Triangulated graph $\mathcal{G}_t = (\mathcal{V}, \mathcal{E} \cup \mathcal{E}_t)$
 1: for each $V \in \mathcal{V}$ do
 2:     $N(V) := Neighbor(V)$
 3: end for
 4: $\mathcal{E}_t := \emptyset$,
 5: for $i = 1$ to $i = |\mathcal{V}|$ do
 6:     for each $X, Y \in N(V_i)$ and $(X, Y) \notin \mathcal{E} \cup \mathcal{E}_t$ do
 7:         $\mathcal{E}_t := \mathcal{E}_t \cup (X, Y)$
 8:         $N(X) := N(X) \cup Y$ and $N(Y) := N(Y) \cup X$
 9:     end for
10:     for each $U \in N(V_i)$ do
11:         $N(U) := N(U) \setminus V_i$
12:     end for
13: end for
```

Figure 2: Triangulating a primal graph, given an arbitrary vertex ordering

to the edge set $\mathcal{E}$. Then a primal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of the CBI problem is constructed.

4.1.2 *Triangulation of Primal Graph*

A graph is triangulated if every cycle of length at least four contains a chord, that is, two non-consecutive vertices on the cycle are adjacent. Triangulated graphs are also called chordal graphs due to the existence of a chord in every cycle. The triangulated graph of graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is obtained by adding edges $\mathcal{E}_t$ to $\mathcal{G}$ such that $\mathcal{G}_t = (\mathcal{V}, \mathcal{E} \cup \mathcal{E}_t)$ is a triangulated graph.

Given an arbitrary vertex ordering $\sigma = (V_1, \cdots, V_n)$, a primal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ can be triangulated using to the procedure described in Figure 2 [22]. Practically, various heuristics are used to find a vertex ordering that optimizes the triangulation. Details of heuristic triangulation algorithms and their empirical evaluations can be found in [15].

4.1.3 *Identifying JT from Triangulated Graph*

Given a triangulated graph, junction tree is constructed from identifying clusters and separators from the triangulated graph. It can be done with searching fully connected components, or cliques. Each clique that is not contained by another clique is corresponding to a cluster $C_i \in \mathcal{C}$. For any two clusters $C_i, C_j \in \mathcal{C}$ and $C_i \cap C_j \neq \emptyset$, there is a separator $S_k \in \mathcal{S}$ and $S_k = C_i \cap C_j$. Then a junction tree $\mathcal{T} = (\mathcal{C}, \mathcal{S})$ for the CBI problem $\mathcal{P}$ is constructed.

4.2 JT Consistency Enforcing

In general, junction tree algorithms assign constraints to clusters (called local constraints). The local constraints are combined and marginalized as a message. The message is passed between clusters. Once all clusters receive messages from all the other clusters, either directly or through the filtering of some other clusters, a partial solution that is guarantied to a part of a solution can be computed at **any** cluster by combining its local constraint and all the incoming messages to it. In other words, the solution can be extended using non-backtrack search from any cluster, if the junction tree is consistent.

```
Input: A junction tree $\mathcal{T} = (\mathcal{C}, \mathcal{S})$ of a CBI problem
    $(\mathbf{X}, \mathbf{D}, \mathbf{S}, \mathbf{F})$
Output: A consistent junction tree
 1: Attach to each cluster $C_i$ a potential $\phi_{C_i} = \mathbf{1}$
 2: for each $f \in F$ do
 3:     Find a cluster $C_i$ such that $Scope(f) \subseteq C_i$
 4:     $\phi_{C_i} := \phi_{C_i} \otimes f$
 5: end for
 6: Choose $C_r$ as the root of the tree
 7: Inward-Passing($\mathcal{T}, C_r$)
 8: Outward-Passing($\mathcal{T}, C_r$)
```

Figure 3: Generalized junction tree consistency enforcing

Following a specified message-passing scheme, the junction tree reaches consistency. For any two connected clusters C_i and C_j, the messages passing from C_i to C_j, $m(C_i, C_j)$, is to compute and store as follows:

$$m(C_i, C_j) := \bigoplus_{C_i \setminus S_{ij}} (\phi_{C_i} \otimes \bigotimes_{C_l \in N_{i-j}} m(C_l, C_i))$$

where N_{i-j} is the set of neighbor cluster of cluster C_i other than C_j and ϕ_{C_i} is the combination of local constraints assigned to cluster C_i.

Given a junction tree representation $\mathcal{T} = (\mathcal{C}, \mathcal{S})$ of a Open CBI problem $\mathcal{P}$, the message-passing to reach consistency is usually organized in two phases: inward phase and outward phase. Inward and outward here is related to which cluster is chosen as the root of the tree. The choice of root is arbitrary and does not violate the correctness of junction tree algorithm. The JT enforcing algorithm is described in Figure 3 with the procedures for inward and outward phase message-passing shown in Figures 4 and 5, respectively.

The soundness and completeness of junction tree algorithms for probabilistic reasoning are proven in [27]. The correctness of generalized junction tree for CBI problems is discussed in [4]. Reader may notice already that the message computing in the outward phase (Line 5) in Figure 5 has room for improvement. The idea here is to cache the computation result in the cluster if the combination operator $\otimes$ supports some special properties. We discussed it in [4] and gave variant JT algorithms when $\otimes$ is idempotent, i.e. logic AND in CSP and SAT and $\otimes$ is invertible, i.e. plus and minus, times and divides.

It is known that both time and space complexities of the junction tree consistency enforcing are in $O(|\mathcal{C}| \cdot d^{w+1})$, where d is the maximal domain size and w is the width of a junction, with treewidth as the lower bound. A junction tree algorithm

```
Input: A junction tree $\mathcal{T} = (\mathcal{C}, \mathcal{S})$ and a root cluster $C_r$
1: for each $C_i \in Child(C_r)$ do
2:     Inward-Passing($\mathcal{T}, C_i$)
3:     $\mathbf{C_c} := Child(C_i)$
4:     $m(C_i, C_r) := \bigoplus_{C_i \setminus S_{ir}} (\phi_{C_i} \otimes \bigotimes_{C_l \in \mathbf{C_c}} m(C_l, C_i))$
5: end for
```

Figure 4: Procedure Inward-Passing($\mathcal{T}, C_r$) for inward phase message-passing

is generally seen as a linear algorithm with a constant factor that is exponential in the treewidth. The success of junction tree algorithms depends on the structure of the given problem. For the Open CBI problems discussed in this paper, the near-optimal junction tree can be constructed off-line, which favors the junction tree algorithm if the structure of the problem is loosely coupled.

4.3 Maintain Consistency with Domain Changes

There are three reasons that make the junction tree representation a suitable graphical model for the Open CBI problems:

1. The variable that changes its domain values induces a subtree possibly smaller than the original junction tree. The satisfiability or the optimal weight can be decided in constant time (for value removal) or time that is linear in the size of the induced subtree (for value addition).

2. For each message that needs to update, only a fraction of tuples with changed values needs to be either appended or removed. This is good for speeding up the message updating.

3. Maintaining the consistency requires theoretically a inward phase of message-passing for the subtree induced by the changing variable and a completed outward phase of message-passing for the whole junction tree. However, the outward phase of message-passing can be terminated at a branch once an invariant message is detected.

4.3.1 Consistency Maintenance with Value Removal

Given a consistent junction tree $\mathcal{T} = (\mathcal{C}, \mathcal{S})$ for $CBI(i)$ and a variable V, which domain is $\mathbf{D_V}(i+1)$ with $\mathbf{D_V}(i+1) \subset \mathbf{D_V}(i)$, all clusters of the junction tree are divided into two parts: (1) clusters that contain V, which is a subtree $\mathcal{T}_\mathcal{V}$ of $\mathcal{T}$ induced by V; and (2) clusters that do not contain V, which is a set of subtrees of $\mathcal{T}$ rooted at leaf clusters of the subtree $\mathcal{T}_\mathcal{V}$ induced by V.

The consistency of $\mathcal{T}_\mathcal{V}$, the subtree induced by V, is not affected by removing some domain values from V. The difference between $CBI(i+1)$ and $CBI(i)$ is that the constraint tuples or variable assignments that contain removed values are no longer available to be a part of the solution. Other tuples, however, are still consistent and could participate in

```
Input: A junction tree $\mathcal{T} = (\mathcal{C}, \mathcal{S})$ and a root cluster $C_r$
1: for each $C_i \in Child(C_r)$ do
2:     $C_p := Parent(C_r)$
3:     $\mathbf{C_c} := Child(C_r)$
4:     $m(C_r, C_i) :=$
5:       $\bigoplus_{C_r \setminus S_{ir}} (\phi_{C_r} \otimes m(C_p, C_r) \bigotimes_{C_j \in \mathbf{C_c}, j \neq i} m(C_j, C_r))$
6:     Outward-Passing($\mathcal{T}, C_i$)
7: end for
```

Figure 5: Procedure Outward-Passing($\mathcal{T}, C_r$) for outward phase message-passing

the solution. So we only need to remove tuples that contains removed values from local constraints and messages in both directions in $\mathcal{T}_\mathcal{V}$. Also we notice that the incoming messages from the second part contain no information of V at all and will not change from $CBI(i)$ to $CBI(i+1)$. The satisfiability or the optimal weight can be solely answered by the consistent $\mathcal{T}_\mathcal{V}$ for $CBI(i+1)$. This is can be done in a single cluster of $\mathcal{T}_\mathcal{V}$. To find the total assignment, however, we need go through the whole junction tree using non-backtrack search after making it consistent.

Clusters in the second part are organized in several subtrees rooted at leaf clusters of $\mathcal{T}_\mathcal{V}$. For each of these subtrees, inward messages from child clusters to parent clusters remain the same from $CBI(i)$ to $CBI(i+1)$. So the inward phase message-passing for this subtree is not required. However, the root of the subtree, the only cluster that contains V in this subtree, is revised because of V's value removal. The outward message-passing is required to make it consistent. We observe that all local constraints and messages in this subtree, except the root cluster, contain no V in their scopes. It means the recursion can be terminated at the branch if we find any unchanged outward message.

The procedure that maintains the consistency of a junction tree while domain values are removed from a variable is shown in Figure 6. The sub-procedure **Outward-Passing** is revised based on Figure 5 with an additional unchanged message detection. It shows that determining satisfiability or the optimal weight requires constant time. We can use cluster size and number of child clusters as heuristics to pick up a root cluster C_r. Enforcing the consistency requires the time that is linear in the fractional size of the tree.

4.3.2 Consistency Maintenance with Value Addition

Given a consistent junction tree $\mathcal{T} = (\mathcal{C}, \mathcal{S})$ for $CBI(i)$ and a variable V, which domain is $\mathbf{D_V}(i+1)$ with $\mathbf{D_V}(i) \subset \mathbf{D_V}(i+1)$, all clusters of the junction tree are divided into two parts: (1) clusters that contain V, which is a subtree $\mathcal{T}_\mathcal{V}$ of $\mathcal{T}$ induced by V; and (2) clusters that do not contain V, which is a set of subtrees of $\mathcal{T}$ rooted at leaf clusters of the subtree $\mathcal{T}_\mathcal{V}$ induced by V.

The consistency of $\mathcal{T}_\mathcal{V}$, the subtree induced by V, is affected by adding some domain values to V. Both inward and out-

```
Input: A junction tree $\mathcal{T} = (\mathcal{C}, \mathcal{S})$ of a $CBI(i)$ and Variable $V$ with domain value removed
Output: A consistent junction tree for $CBI(i+1)$
 1: Let $\mathcal{T}_\mathcal{V} = (\mathcal{C}_\mathcal{V}, \mathcal{S}_\mathcal{V})$ be the subtree induced by $V$
 2: Pick a cluster $C_r \in \mathcal{C}_\mathcal{V}$ as the root of both $\mathcal{T}_\mathcal{V}$ and $\mathcal{T}$
 3: Remove tuples contain $V$ from $\phi_{C_r}$
 4: for each $C_i \in Child(C_r)$ do
 5:     Remove tuples contain $V$ from $m(C_i, C_r)$
 6: end for
 7: %Answer the satisfiability or the optimal weight
 8: $\mathbf{C_c} := Child(C_r)$
 9: Report $\bigoplus_{C_r} (\phi_{C_r} \otimes \bigotimes_{C_i \in \mathbf{C_c}} m(C_i, C_r))$
10: % Consistency enforcing
11: for each $C_i \in \mathcal{C}_\mathcal{V}$ do
12:     Remove tuples contain $V$ from $\phi_{C_r}$
13: end for
14: for each $S_{ij} \in \mathcal{S}_\mathcal{V}$ do
15:     Remove tuples contain $V$ from $m(C_i, C_j)$
16:     Remove tuples contain $V$ from $m(C_j, C_i)$
17: end for
18: for each $C_i \in \mathcal{C}_\mathcal{V}$ and $C_i$ is a leaf cluster of $\mathcal{T}_\mathcal{V}$ do
19:     Let $\mathcal{T}_{C_i}$ be the subtree of $\mathcal{T}$ rooted at $C_i$
20:     Outward-Passing($\mathcal{T}_{C_i}, C_i$)
21: end for
```

Figure 6: JT consistency maintenance while some domain values are removed from a variable

ward message-passing phases are required to make it consistent. However, for local constraints and messages in $\mathcal{T}_\mathcal{V}$, tuples without new values remain the same from $CBI(i)$ to $CBI(i+1)$. Only tuples contain new values are appended to local constraints and messages in $CBI(i+1)$. That makes both inward and outward message-passing computation less complicated than the original procedures introduced in Figures 4 and 5. Once the inward phase is done for $\mathcal{T}_\mathcal{V}$, the satisfiability or the optimal weight can be solely answered by the local constraint and all incoming messages of the root cluster C_r for $CBI(i+1)$. It is the same as the case of value removal. Heuristics can be used here to find a good root cluster. Also, to find the total assignment, we need go through the whole junction tree using non-backtrack search after making it consistent.

To maintain consistency for clusters in the second part, organized in several subtrees rooted at leaf clusters of $\mathcal{T}_\mathcal{V}$, the same operation is required as we did in the case of value removal. No inward phase is required for these subtrees. Only outward phase is needed and the recursion can be terminated at a branch if an unchanged message is detected.

The procedure that maintains the consistency of a junction tree while domain values are added to a variable is shown in Figure 7. Both the sub-procedures, **Inward-Passing** and **Outward-Passing**, are revised based on Figure 4 and Figure 5, respectively, with additional unchanged message detection. Both of them compute and append only new tuples

```
Input: A junction tree $\mathcal{T} = (\mathcal{C}, \mathcal{S})$ of a $CBI(i)$ and Variable $V$ with domain value added
Output: A consistent junction tree for $CBI(i+1)$
 1: Let $\mathcal{T}_\mathcal{V} = (\mathcal{C}_\mathcal{V}, \mathcal{S}_\mathcal{V})$ be the subtree induced by $V$
 2: Pick a cluster $C_r \in \mathcal{C}_\mathcal{V}$ as the root of both $\mathcal{T}_\mathcal{V}$ and $\mathcal{T}$
 3: for each $C_i \in \mathcal{C}_\mathcal{V}$ do
 4:     Update $\phi_{C_i}$ with tuples contain new values
 5: end for
 6: Inward-Passing($\mathcal{T}_\mathcal{V}, C_r$)
 7: %Answer the satisfiability or the optimal weight
 8: $\mathbf{C_c} := Child(C_r)$
 9: Report $\bigoplus_{C_r} (\phi_{C_r} \otimes \bigotimes_{C_i \in \mathbf{C_c}} m(C_i, C_r))$
10: % Consistency enforcing
11: Outward-Passing($\mathcal{T}_\mathcal{V}, C_r$)
12: for each $C_i \in \mathcal{C}_\mathcal{V}$ and $C_i$ is a leaf cluster of $\mathcal{T}_\mathcal{V}$ do
13:     Let $\mathcal{T}_{C_i}$ be the subtree of $\mathcal{T}$ rooted at $C_i$
14:     Outward-Passing($\mathcal{T}_{C_i}, C_i$)
15: end for
```

Figure 7: JT consistency maintenance while some domain values are added to a variable

with added values. It shows that answering the satisfiability or the optimal weight requires the time that is linear in the size of the subtree induced by V. Enforcing the consistency requires the time that is linear in the fractional size of the tree.

5. CONCLUSION AND FUTURE WORK

Open constraint programming [12], including open constraint satisfaction (Open CSP) and open constraint optimization (Open COP), is an extended constraint programming framework to model and solve practical applications with open-world settings. It assumes that domain values are incrementally discovered and the values are discovered in a strict non-decreasing order of cost if optimization is in the consideration. The open constraint programming has been proven to be a suitable framework to model many real world application, especially those inspired by the increasing use of the Internet. The assumptions associated with the open constraint programming, however, impose limitations on modelling some problems that fit the constraint programming paradigm and have open-world settings. For example, some revealed domain values may become unavailable due to the changing circumstances of the agent who is the owner of the variable. The assumption of incrementally discovered domain values does not always hold in this case. The owner of a variable may not want to reveal its top preference values to the solver because of privacy concerns or cannot reveal them because they are not available at the time of consideration. In some applications, the request for new value sending from the solver cannot always be fulfilled. That violates the assumption of discovering values in a non-decreasing order of cost. Also it might be the owner of variables who reports value changes to the solver. In the Open CSP and Open COP, however, it is the solver who initials the request for new values and the request is always fulfilled immediately. These limitations inspired us proposing an Open Constraint-Based Inference (Open CBI) framework as an extension to Open Constraint Programming.

As the first contribution of this paper, the proposed Open CBI framework is based on the unified semiring-based CBI framework [4]. By using various semirings, the Open CBI framework is shown to be a superset of Open COP. Also the CBI framework can be used to model open constraint programming problems with both domain value discovery and removal and non-monotonic revealing of domain values with regard to the cost or preference. An Open CBI problem is modelled as a sequence of CBI problems. A CBI problem changes to the next one in the sequence when a variable reports the domain change to the solver. The extensibility of the Open CBI framework is ensured by using the semiring concept to generalize various problems in different research fields.

The junction tree representation is a widely studied graphical model to maintain knowledge and facilitate computation in the probabilistic reasoning. We have shown that the junction tree representation and junction tree algorithms can be generalized within the CBI framework. As the second contribution of this paper, we extend the junction tree representation to handle Open CBI problems. We show in this paper that the junction tree representation is a suitable graphical model to reuse the computed intermediate constraints to subproblems. According to the definition of Open CBI, the structure of the problem is unchanged, thus we can construct off-line the near optimal junction tree. By using the provided message-passing procedures to make the junction tree consistent, satisfiability or the optimal weight can be answered within a local cluster and its incoming messages. Finding out the total assignment requires going through the whole junction tree, but it can be done using non-backtrack search on the fly.

We also proposed a junction tree consistency maintenance algorithm for Open CBI problems with domain value changes. This is the third contribution of this paper. We analyze and discuss that both determining satisfiability or the optimal weight and finding total assignment of variables have time complexity that is linear in the size of the junction tree in time possibly less than the time required to enforce the consistency from scratch. We are doing experiments to evaluate the effectiveness of these algorithms based on the GCBIJ toolkit [4] and plan to report it in subsequent work.

The junction tree algorithms, in general, are linear in the size of junction tree. Both the time and space complexities contain a constant factor that is exponential in the maximum cluster size of the tree. Treewidth is the lower bound of the width of all possible junction tree representations of a given Open CBI problem. The proposed junction tree representation and algorithms are suitable for an Open CBI problem with a tractable treewidth. For a Open CBI problem with a large treewidth, other graphical models such as junction graph with smaller cluster size can be options to replace the junction tree representation. Initial work on this direction has been done in [4]. We can also import arc consistency [19] and its non-binary version, generalized arc consistency [20, 21], as candidate inference algorithms in future work. These algorithms have been applied to CBI problems with closed-world settings and received some preliminary success [5, 6]. Stronger local consistencies, such as directional arc consistency [7], full directional arc consistency [17] and existential arc consistency [9], as well as Soft Arc Consistency [7] offer the potential to be integrated into the Open CBI framework. Finally, we will also investigate the possibilities of combining together backtrack search and inference approaches.

6. ACKNOWLEDGMENTS

We thank the anonymous reviewers for their comments on this paper. This research was supported by the Natural Sciences and Engineering Research Council of Canada. Alan K. Mackworth holds a Canada Research Chair.

7. REFERENCES

[1] Srinivas M. Aji and Robert J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46:325–343, 2000.

[2] Stefan Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic and Discrete Methods*, 8:277–284, 1987.

[3] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, 1997.

[4] Le Chang. Generalized constraint-based inference. Master's thesis, University of British Columbia, 2005. www.cs.ubc.ca/grads/resources/thesis/May05/Le_Chang.pdf.

[5] Le Chang and Alan K. Mackworth. A generalization of generalized arc consistency: From constraint satisfaction to constraint-based inference. In *IJCAI05 Workshop on Modelling and Solving Problems with Constraints*, pages 68–75, Edinburgh, July 2005.

[6] Le Chang and Alan K. Mackworth. Constraint-based inference using local consistency in junction graphs. In *ECAI2006 Workshop on Inference Methods Based on Graphical Structures of Knowledge*, pages 1–6, Riva del Garda, Italy, August 2006.

[7] Martin Cooper and Thomas Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154(1-2):199–227, 2004.

[8] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill, 1990.

[9] Simon de Givry, Matthias Zytnicki, Federico Heras, and Javier Larrosa. Existential arc consistency: Getting closer to full arc consistency in weighted csps.

In *Proceedings of IJCAI-05*, Edinburgh, Scotland, 2005.

[10] Rina Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *12th Conf. on Uncertainty in Artificial Intelligence*, pages 211–219, 1996.

[11] Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artif. Intell.*, 38(3):353–366, 1989.

[12] Boi Faltings and Santiago Macho-Gonzalez. Open constraint programming. *Artif. Intell.*, 161(1-2):181–208, 2005.

[13] F. V. Jensen, S. L. Lauritzen, and K. G. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 4:269–282, 1990.

[14] Kalev Kask, Rina Dechter, Javier Larrosa, and Avi Dechter. Unifying cluster-tree decompositions for reasoning in graphical models. *Artificial Intelligence*, 166:165–193, August 2005.

[15] Uffe Kjærulff. *Aspects of Efficiency Improvement in Bayesian Networks*. PhD thesis, Aalborg University, 1993.

[16] J. Kohlas and P.P. Shenoy. Computation in valuation algebras. In *Handbook of Defeasible Reasoning and Uncertainty Management Systems, Volume 5: Algorithms for Uncertainty and Defeasible Reasoning*, pages 5–40. Kluwer, Dordrecht, 2000.

[17] Javier Larrosa and Thomas Schiex. In the quest of the best form of local consistency for weighted CSP. In *Proc. of IJCAI-03*, pages 239–244, Acapulco, Mexico, 2003.

[18] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B*, 50:157–224, 1988.

[19] Alan K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[20] Alan K. Mackworth. On reading sketch maps. In *IJCAI77*, pages 598–606, 1977.

[21] Roger Mohr and G. Masini. Good old discrete relaxation. In *European Conference on Artificial Intelligence*, pages 651–656, 1988.

[22] Richard E. Neapolitan. *Probabilistic Reasoning in Expert Systems: Theory and Algorithms*. John Wiley and Sons, New York, NY, 1990.

[23] Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, pages 266–271, 2005.

[24] Adrian Petcu and Boi Faltings. ODPOP: An algorithm for open/distributed constraint optimization. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-06*, pages 703–708, Boston, USA, July 2006.

[25] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Foundations of Artificial Intelligence. Elsevier Science Publishers, Amsterdam, The Netherlands, 2006.

[26] Thomas Schiex, Hélène Fargier, and Gerard Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In *IJCAI95*, pages 631–637, Montreal, 1995.

[27] P. P. Shenoy and G. Shafer. Axioms for probability and belief-function propagation. In *UAI90*, pages 169–198. 1990.

[28] Y. Zhang and Alan Mackworth. Parallel and distributed finite constraint satisfaction: Complexity,algorithms and experiments. In V. Kumar L. Kanal, H. Kitano and C. Suttner, editors, *Parallel Processing for Artificial Intelligence*, volume 1, pages 305–334. Elsevier, Amsterdam, 1994.