Properties of Tree Convex Constraints [1,2]

Yuanlin Zhang [a,*] Eugene C. Freuder [b]

[a] *Department of Computer Science, Texas Tech University, Lubbock, USA*

[b] *Cork Constraint Computation Center, University College Cork, Ireland*

Abstract

It is known that a tree convex network is globally consistent if it is path consistent. However, if a tree convex network is not path consistent, enforcing path consistency on it *may not* make it globally consistent. In this paper, we investigate the properties of some tree convex constraints under intersection and composition. As a result, we identify a sub-class of tree convex networks that are *locally chain convex and strictly union closed*. This class of problems can be made globally consistent by arc and path consistency and thus is tractable. Interestingly, we also find that some scene labeling problem can be modeled by tree convex constraints in a natural and meaningful way.

Key words:
Constraint networks, tree convex constraints, row convex constraints, connected row convex constraints, scene labeling problem.

1 Introduction

A binary constraint network is tree convex [20] if we can construct a tree for the domain of each variable so that for any constraint, no matter what value one variable takes, all the values allowed for the other variable form a subtree of the constructed tree. As an example, the constraint c_{xy} of Fig. 1(a) is tree convex while c'_{xy} of Fig. 1(c) is not.

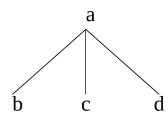
* Corresponding author.
Email addresses: `yzhang@cs.ttu.edu` (Yuanlin Zhang), `e.freuder@4c.ucc.ie` (Eugene C. Freuder).

[1] A preliminary version of this paper appeared in [18].

[2] This work has received support from Science Foundation Ireland under Grant 00/PI.1/C075.

$$c_{xy} = \begin{array}{c} a\,b\,c\,d \\ \begin{bmatrix} 1110 \\ 1011 \\ 1101 \end{bmatrix} \end{array} \begin{array}{c} a \\ b \\ c \end{array}$$

(a)



(b)

$$c'_{xy} = \begin{bmatrix} 110 \\ 011 \\ 101 \end{bmatrix}$$

(c)

Fig. 1. **(a)** Constraint c_{xy} is represented by a matrix. The column $\{a, b, c\}$ beside the matrix is the domain of x, and the row $\{a, b, c, d\}$ above the matrix is the domain of y. **(b)** A tree constructed for the values of the domain of y. c_{xy} is tree convex with respect to this tree. **(c)** c'_{xy} is obtained from c_{xy} by deleting the value a from the domain of y. c'_{xy} is not tree convex with respect to any tree.

Tree convex constraints further the study of the convexity and monotonicity of constraints [14,13]. It has been shown that a tree convex network is globally consistent if it is path consistent. However, if a tree convex network is not path consistent, enforcing path consistency on it *may not* make it globally consistent because some constraints may be modified during the enforcing procedure and thus may no longer be tree convex.

In this paper, we examine the tree convex constraints and characterize conditions under which the desirable tree convex property of a network is preserved when arc and path consistency are enforced. We then identify a tractable class of restricted tree convex constraints. This result generalizes the earlier work on monotone [13] and connected row convex constraints [5]. The latter is built on the work of [14]. Finally, we show that tree convex constraints help to model some scene labeling problems in a natural and meaningful way. The related work by Jeavons et al. [7,8] and Kumar [10] will be discussed in the last section.

2 Preliminaries

In this section, we review the basic concepts and notations used in this paper.

Constraint Networks A binary *constraint network* consists of a set of variables $V = \{x_1, x_2, \cdots, x_n\}$ with a finite domain D_i for each variable $x_i \in V$, and a set of binary constraints C over the variables of V. c_{xy} denotes a constraint on variables x and y which is defined as a relation over D_x and D_y. Operations on relations, e.g., *intersection* ($\cap$)*, composition* ($\circ$)*, and inverse*, are applicable to constraints.

We assume that, between any ordered variables (x, y), there is only one constraint. *c_{xy} and c_{yx} are considered to be two different constraints*. However, we assume the inverse of c_{xy} is equal to c_{yx}.

Image Given a constraint c_{xy} and a value $u \in D_x$, $v \in D_y$ is a *support* of u if u and v satisfy c_{xy}, that is $(u, v) \in c_{xy}$. The *image* of u under c_{xy}, denoted by $I_y(u)$, is the set of all its supports in D_y. The *image of a subset* of D_x is the union of the images of its values.

k-consistency A constraint network is *k-consistent* if any consistent instantiation of any distinct $k-1$ variables can be consistently extended to any new variable. A network is *strongly* k-consistent if it is j-consistent for all $j \leq k$. A strongly n-consistent network is called *globally consistent*. 2- and 3-consistency are usually called *arc consistency* and *path consistency* respectively. Note that, under this definition, we need to add a universal constraint between variables that are not explicitly constrained by the network.

More materials on these concepts can be found in [11,13,6].

Forests, Trees, Chains, and Sets In the following we review trees that play a fundamental role in the analysis of tree convex constraints and introduce some new notations used in this paper. A *forest* is a graph without any cycles. A *tree* is a connected graph without any cycles. A forest can be regarded as a set of trees. In the rest of the paper, *we always assume there is a root for a tree in a forest*. The path between any two nodes (or vertices) of a tree is unique and the *distance* of a node to the root is defined as the number of edges in the path between them. Given a tree, a *subtree* is defined as a connected subgraph of the tree, and its *root* is the node closest to the root of the tree.

A *forest on a set S* is a forest whose vertex set is exactly S. We also call a set I a *subtree* of a forest $\mathcal{T}$ if there exists a subtree of some tree in $\mathcal{T}$ such that its vertex set is exactly I. An empty set is a *subtree* of any forest. A tree (and subtree respectively) becomes a *chain* (and *subchain* respectively) if each of its nodes has at most one child. The *last value (or node)* of a subchain is the farthest one away from its root. For example, the graph in Fig. 1(b) is a tree on $\{a, b, c, d\}$. $\{a, b, c\}$ is a subtree of it, and $\{a, b\}$ is a subchain whose last value is b.

The *intersection* of two trees is defined as the graph whose vertices and edges are in both trees. It has the following property:

Proposition 1 [20] *Let T_1, T_2 be two subtrees of some tree. The intersection of T_1 and T_2 is also a subtree of the tree. Furthermore, if the intersection is not empty, the root of the intersection is either the root of T_1 or that of T_2.*

Next, we relax the tree structures, used in some concepts in [20], to the forest structures.

Definition 1 *Sets* $E_1, \cdots, E_k$ *are* tree convex with respect to a forest $\mathcal{T}$ on $\bigcup_{i \in 1..l} E_i$ *if every* E_i *is a subtree of* $\mathcal{T}$.

For example, given the tree in Fig. 1(b), sets $\{a, b, c\}$, $\{a, b, d\}$, and $\{a, c, d\}$ are tree convex.

Definition 2 *A constraint* c_{xy} *is* tree convex with respect to a forest $\mathcal{T}$ on D_y *if the images of all values in* D_x *are tree convex with respect to* $\mathcal{T}$.

Example Consider c_{xy} in Fig. 1(a). The images of a, b, c are $\{a, b, c\}$, $\{a, c, d\}$, and $\{a, b, d\}$ respectively. They are tree convex with respect to the tree in Fig. 1(b) and thus c_{xy} is tree convex with respect to that tree. The readers are invited to verify that there is no tree to make c'_{xy} (in Fig. 1(c)) tree convex. □

In [20], a tree convex constraint network is defined as a network where all constraints are tree convex with respect to a common tree on *the union of all domains* in the network. In the following definition, only the forests on the *individual* domains matter.

Definition 3 *A constraint network is* tree convex *if there exists a forest on the domain of each variable such that every constraint* c_{xy} *of the network is tree convex with respect to the forest on* D_y.

As pointed out by one of the referees, the new definition of tree convexity of constraint networks is equivalent to the old ones [20] if the domains of the variables are disjoint. Given any problem, we can make the domains of the variables disjoint by renaming the values of the domains of the variables so that they are different from those of the domains of the other variables. The renaming preserves the solutions of a constraint network.

One advantage of the new definition is that even if the domains of two variables share some values, it explicitly allows us to construct different forests for them in deciding the tree convexity of the network. More importantly, in this paper, we need to introduce further restrictions (e.g., consecutiveness) on tree convex constraints. The forest-based definition helps to simplify the presentation, the proofs, and the understanding of the results.

The tree convex set intersection lemma in [20] still holds for the new definition of tree convex sets, which can be lifted to the following consistency result.

Proposition 2 *A tree convex constraint network is globally consistent if it is path consistent.*

The proof follows directly from that of [20] because the new definition does not affect the essential part of that proof.

3 Properties of Intersection and Composition of Tree Convex Constraints

A network can be made path consistent by removing from the constraints the tuples which can not be consistently extended to a new variable. It is equivalent to the matrix computation $c_{xy} = c_{xy} \cap (c_{zy} \circ c_{xz})$, where $\circ$ denotes composition. To make use of Theorem 2, we need to study the impact of the intersection and composition operations on the tree convexity of constraints.

Intersection preserves tree convexity.

Proposition 3 *Assume constraints c^1_{xy} and c^2_{xy} are tree convex with respect to a forest $\mathcal{T}$ on the domain D_y. Their intersection is also tree convex.*

Proof. Let $c_{xy} = c^1_{xy} \cap c^2_{xy}$. For any $v \in D_x$, its images under c^1_{xy} and c^2_{xy} are both subtrees of $\mathcal{T}$. The intersection of the two images is a subtree of $\mathcal{T}$ by Proposition 1. That is, the image of every $v \in D_x$ is a subtree of $\mathcal{T}$. Hence, c_{xy} is tree convex. □

However, the composition of tree convex constraints might not preserve the tree convexity. Let us use a more intuitive way than matrix multiplication to understand the composition. Consider the constraints in Fig. 2. After composing c_{xy} and c_{yz}, the image of a under the composition c_{xz} is $\{a, b, c, d\}$ that is exactly the union of the images of b and d in D_y under c_{yz}. To assure that the image of a under c_{xz} is a tree, we can simply require that $I_z(b) \cup I_z(d)$ is a (sub)tree, that is $I_z(b)$ and $I_z(d)$ touch each other.

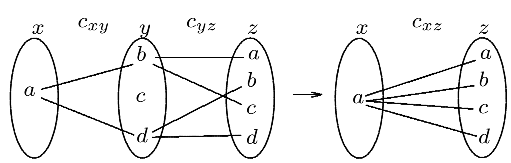


Fig. 2. The composition of two constraints. In the diagrams of this paper, a value is drawn as a dot or letter, and a variable is drawn as an ellipse. The values inside an ellipse form the domain of the corresponding variable. The edges between two ellipses specify the constraint between the corresponding variables.

Definition 4 *A tree convex constraint c_{xy} with respect to a forest $\mathcal{T}_y$ on D_y is* consecutive *with respect to a forest $\mathcal{T}_x$ on D_x if and only if for every two neighboring values a, b on $\mathcal{T}_x$, $I_y(a) \cup I_y(b)$ is a subtree of $\mathcal{T}_y$. A constraint network is* tree convex and consecutive *iff there exists a forest on each domain such that every constraint c_{xy} is tree convex and consecutive with respect to the forests on D_y and D_x.*

Proposition 4 *The class of consecutive tree convex constraints is closed under composition.*

Proof. Let c_{xy} and c_{yz} be two consecutive tree convex constraints with respect to forests $\mathcal{T}_x$, $\mathcal{T}_y$ and $\mathcal{T}_z$ on D_x, D_y and D_z respectively, and c_{xz} the composition of c_{xy} and c_{yz}. Firstly, we show that c_{xz} is tree convex. Consider any $v \in D_x$. Let its image in D_y be $I_y(v)$. The image of v under c_{xz} would be $\cup_{b \in I_y(v)} I_z(b)$ where $I_z(b)$ is the image of b under c_{yz}. Since the union of the images of any neighboring values in $I_y(v)$ is a subtree of $\mathcal{T}_z$, the union of all the images of values of $I_y(v)$ is a subtree of $\mathcal{T}_z$.

Secondly, we show that c_{xz} is consecutive. Let $u, v \in D_x$ be neighbors under $\mathcal{T}_x$. Let $I_z(u)$ and $I_z(v)$ be their images under c_{xz}. Since c_{xy} is consecutive, $I_y(u) \cup I_y(v)$ is a subtree of $\mathcal{T}_y$. Hence, the union of the images (with respect to c_{yz}) of the values of $I_y(u) \cup I_y(v)$ is a subtree of $\mathcal{T}_z$ due to the consecutiveness of c_{yz}. Therefore, $I_z(u) \cup I_z(v)$ is a subtree of $\mathcal{T}_z$. $\Box$

4 Tractable Tree Convex Constraint Networks

The intersection of two subtrees may be an empty set, which means that, after the intersection of two tree convex constraints, the image of a value could be empty. Deleting such a value could make a constraint no longer tree convex, which is shown by the example in Fig. 1. It is also interesting to note that a constraint c_{xy} may become tree convex after a sufficient number of values are removed from D_y.

The following special class of tree convex constraints that is closed under the operation of deleting values.

Definition 5 *A constraint c_{xy} is* locally chain convex *with respect to a forest on D_y if and only if the image of every value in D_x is a subchain of the forest. A constraint network is* locally chain convex *iff there exists a forest on each domain such that every constraint c_{xy} is locally chain convex with respect to the forest on D_y.*

For example, under the tree for D_y in Fig. 1(b), the constraint in Fig. 1(a) is not locally chain convex because the image of $a \in D_x$ is $\{a, b, c\}$ that is not a subchain of the tree on D_y. In fact, there does not exist any tree to make it chain convex.

Proposition 5 *A locally chain convex constraint network (V, D, C) is still locally chain convex after the removal of any value from any domain.*

Proof. Assume the forest on D_y is $\mathcal{T}_y$ and a value v is removed from D_y. The removal of v does not affect the property of any constraint $c_{yx} \in C$. We need to show that every $c_{xy} \in C$ is locally chain convex. The deletion of v could make the images of some values of D_x not connected. By constructing a new forest $\mathcal{T}_y''$ on D_y, those broken subchains would be connected under $\mathcal{T}_y''$. Let the children of v be $v_1, \cdots, v_l$ and the parent of v be p_v. Construct a new forest $\mathcal{T}_y'$ from $\mathcal{T}_y$ by removing v and all edges incident on v. If v is the root of some tree of $\mathcal{T}_y$, let $\mathcal{T}_y''$ be $\mathcal{T}_y'$. The image of any value a of D_x either contains v or not. In the latter case, the image is still a chain. In the former case, v is the shallowest node of the image, a chain, and thus the image is still a chain after the removal of v. If v is not the root of any tree of $\mathcal{T}_y$, construct $\mathcal{T}_y''$ from $\mathcal{T}_y'$ by adding an edge between p_v and v_i for all $i (1 \leq i \leq l)$. The image of any value a of D_x is a subchain of $\mathcal{T}_y''$. □

To identify a tractable class of tree convex constraints, a first attempt is to combine the local chain convexity (for deleting a value) with consecutiveness (for composition). However, the composition may destroy the chain convexity, as shown by the example in Fig. 3(a).

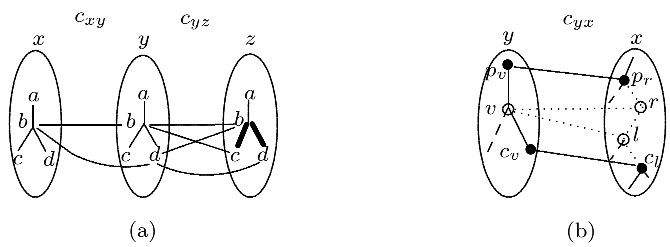


Fig. 3. In this diagram, we draw the tree on a domain inside an ellipse. **(a)** Both c_{xy} and c_{yz} are locally chain convex, but their composition is not because the image of $b \in D_x$ under this composition is $\{b, c, d\}$ (the darkened shape) that is not a subchain. **(b)** t_y contains the solid lines in D_y. t_x contains (p_r, r, l, c_l). t_r contains (r, l).

The image of a value under the composition is the union of several subchains. This union can not be guaranteed to be a subchain by the consecutiveness of the constraints. We need a stronger restriction.

Definition 6 *A constraint* c_{xy} *is* locally chain convex and strictly union closed *with respect to forests* $\mathcal{T}_x$ *on* D_x *and* $\mathcal{T}_y$ *on* D_y *iff the image of any subchain of* $\mathcal{T}_x$ *is a subchain of* $\mathcal{T}_y$.

Remark Local chain convexity and strict union closedness imply consecutiveness of a constraint network; but consecutiveness of a locally chain convex network might not imply strict union closedness as shown by the example in Fig. 3.

Now we introduce the class of constraint networks that is closed under the removal of a value, and the intersection and composition of constraints.

Definition 7 *A constraint network is* locally chain convex and strictly union closed *iff there exists a forest on each domain such that every constraint* c_{xy}

of the network is locally chain convex and strictly union closed with respect to the forests on D_x and D_y.

Theorem 1 *A locally chain convex and strictly union closed constraint network (V, D, C) can be transformed to an equivalent globally consistent network in polynomial time.*

Proof. We show that the given network is locally chain convex after arc and path consistency are enforced on it. In accordance with Theorem 2, the new network is globally consistent. It is known that arc and path consistency enforcing [19] are of polynomial complexity.

Since arc consistency enforcing only removes values from domains, we show that after the removal of any value $v \in D_y$ the network is still locally chain convex and strictly union closed. That is, we show that all constraints $c_{xy}, c_{yx} \in C$ are locally chain convex and strictly union closed.

Case 1. Consider any $c_{xy} \in C$ and the forests $\mathcal{T}_x$ on D_x and $\mathcal{T}_y$ on D_y. Similar to the proof of Proposition 5, we can construct a new forest $\mathcal{T}_y''$ for y such that for every subchain of $\mathcal{T}_x$, its image is still a subchain under $\mathcal{T}_y''$.

Case 2. Consider any constraint $c_{yx} \in C$ and the forests $\mathcal{T}_x$ on D_x and $\mathcal{T}_y$ on D_y. If it is still locally chain convex and strictly union closed, we are done. Otherwise, there exists a subchain t_y of $\mathcal{T}_y$ such that it contains v and its image is no longer a connected graph due to the removal of v. See Fig. 3(b). Let t_x be the image of t_y before removing v. After the removal of v, t_x is broken into two chains. Let the gap (removed subchain) in t_x be t_r. Note t_r might not be equal to the image of v due to the possible overlapping of the image of v and that of its parent and/or child. Let r be the root and l the last node of t_r. Let p_v and p_r be the parents of v and r respectively, and c_v and c_l the children of v and l respectively. Consider any node $u \in t_r$. We know that u is supported by v, but not by p_v or by c_v in t_y. Further, since c_{xy} is locally chain convex and strictly union closed, the image of t_x must be a subchain containing (p_v, v, c_v). It implies that the image of u must be on or contain the subchain (p_v, v, c_v) . Hence, v is the only support of u. After v is gone, u should also be removed. After the removal of t_r, the image of t_y is now connected and thus a subchain.

Next, we show that path consistency enforcing preserves the local chain convexity and strict union closedness. For any constraint c_{xz}, path consistency is usually done by first composing c_{xy} and c_{yz}, and then setting the new constraint between x and z to be the intersection of c_{xz} and $c_{yz} \circ c_{xy}$.

Firstly, we show that the composition of c_{xy} and c_{yz} is locally chain convex and strictly union closed. Assume c_{xy} and c_{yz} are locally chain convex and strictly union closed with respect to the forests $\mathcal{T}_x$, $\mathcal{T}_y$ and $\mathcal{T}_z$ on D_x, D_y and D_z respectively. For any subchain $t_x \in D_x$, its image t'_y under c_{xy} is a subchain.

Since the image of t'_y with respect to c_{yz} is a subchain of D_z, the image of t_x under the composition is a subchain of D_z.

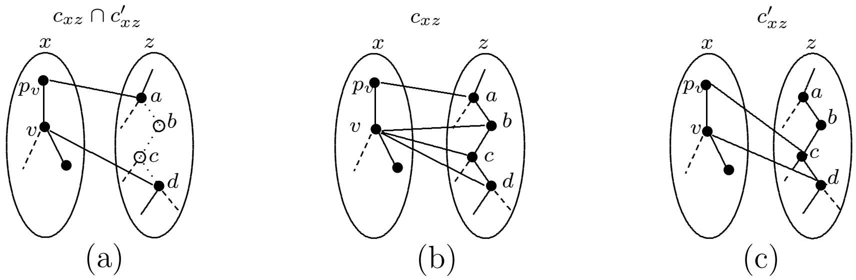


Fig. 4. **(a)** $c_{xz} \cap c'_{xz}$. In the intersection, assume b and c are not shared by the images of v under c_{xz} and under c'_{xz}. The constraints c_{xz} and c'_{xz} should have a form as shown in **(b)** and **(c)**.

Secondly, we show that the intersection, c''_{xz}, of c_{xz} and c'_{xz} $(= c_{xy} \circ c_{yz})$ is locally chain convex and strictly union closed.

Consider a subchain, with only one value, of D_x. Its images under c_{xz} and c'_{xz} are subchains of the forest on D_z. Their intersection is still a chain and thus v's image under c''_{xz} is a subchain.

Consider a subchain t_x, with more than one values, of D_x. In this paragraph, *when we refer to an image, it is under* c''_{xz}. If the image of t_x is a subchain of D_z, we are done. Otherwise, let t''_z be the image of t_x. t''_z is not a subchain. Since the intersection does not form a cycle, t''_z must not be connected. Starting from the root of t_x, we find the first value $v \in t_x$ whose image is disjoint from the image of its parent p_v. Assume the image of v is below that of p_v (the opposite can be proved similarly). Let a be the last value of p_v's image. Let d be the root of v's image. See Fig. 4(a). Let u be any value between (but not including) a and d in D_z. We next prove that there is no support for u. Hence, values between a and d should be removed and the image of t_x is a chain after the deletion.

Let p_v's images under c_{xz} and c'_{xz} be $I(p_v)$ and $I'(p_v)$ respectively. The intersection of $I(p_v)$ and $I'(p_v)$ is a subchain of D_z. Since both $I(p_v)$ and $I'(p_v)$ are chains, a must be the last value of either $I(p_v)$ or $I'(p_v)$. Assume it is the last value of $I(p_v)$. See Fig. 4(b). It implies p_v is not in u's image $I(u)$ under c_{zx}, since u is between a and d. $I(u)$ has to be below p_v (not including it) because $I(u)$ is a chain. Let $I(v)$ and $I'(v)$ be the images of v under c_{xz} and c'_{xz} respectively. $I(v)$ should include at least d and all values between a and d in the tree D_z because c_{xz} is locally chain convex and strictly union closed. Since d is the root of $I(v) \cap I'(v)$, $I'(v)$ includes d but does not include values above d (see Fig. 4(c)). Hence, v is not a support of u (under c'_{xz}), implying that $I'(u)$ has to be above v (not including it). Therefore, the image of u under c''_{xz} is empty because it is the intersection of $I(u)$ and $I'(u)$. In other words, u has no support in the intersection of c_{xz} and c'_{xz}.

Now we are able to discuss a case ignored in the previous discussion. In the original constraint network, there might not be any constraint between some variables, say x and y. Without loss of generality, we assume the graph of the original network is connected. Therefore, there must be a path from x to y. All constraints on the path are locally chain convex and strictly union closed. By the result in the previous paragraphs, the intersection and composition of locally convex and strictly union closed constraints are closed. Let c'_{xy} be the composition of the constraints over the path in order. The constraint c'_{xy} is locally chain convex and strictly union closed. Now, before enforcing path consistency (and possibly one more round of arc consistency), set the constraint between x and y to be c'_{xy} and repeat this for any two variables without a direct constraint on them. After this modification, for any two variables there is a constraint on them that is locally chain convex and strictly union closed. Hence, the constraint network is locally chain convex after enforcing path consistency and thus is globally consistent. □

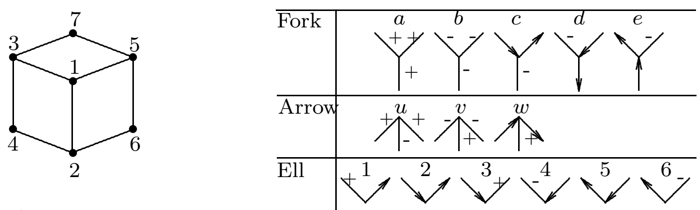
5 An Application of Tree Convex Networks

In this section, we examine the application of tree convex constraints to a scene labeling problem.

Given a two dimensional line drawing of a physical world of plane-faced objects, the scene labeling problem is to identify from the drawing the physical objects and their spatial relations with the requirement that the identification agrees with a human being. Waltz and others reduce this problem into a problem of associating a line with a *label* such that a set of concrete constraints on the *junctions* are satisfied. Given a line drawing, a *junction* is defined as the maximum set of lines that intersect at the same point. Note that *lines* of a drawing correspond to edges of physical objects, and *junctions* correspond to vertexes of physical objects. There are only three types of edges that a line can represent: convex, concave, and boundary edges that are denoted by the *labels* `+`, `-` and `>` respectively. An edge is the intersection of two surfaces of an object. It is *convex* if it can be touched by a ball from the front. For example, when there is a cube in front of a viewer, its top edge is convex for the viewer. An edge is *concave* if it can never be touched by a ball. For example, when a viewer faces a wall and a floor, their intersection edge is concave because there is no way to make a ball touch the edge from the front. A *boundary* edge is the intersection of the background and a surface of an object of concern. An excellent exposition of scene labeling problems can be found in the book [16], and a detailed treatment of this topic can be found in [15].

Scene labeling problems are NP-hard [9]. In the following, we show that some scene labeling instance can be modeled naturally by tree convex constraints and solved efficiently.

Consider the line drawing in Fig. 5 taken from [14]. This drawing involves three types of junctions: Fork, Arrow, and Ell. The shape of junction 1 is a Fork, that of the junctions 3 and 5 is an Arrow, and that of junctions 4, 6 and 7 is an Ell. To label this drawing is to find a solution of a constraint network defined as follows. We introduce a variable x_i for each junction i. A value for a variable is a way to label the lines in the corresponding junction. Under appropriate assumptions, there are only 5 physically realizable ways to label a Fork, 3 an Arrow, and 6 an Ell, which are listed in Fig. 5. The constraints on the variables are straightforward, i.e., any two variables should take the same label on their shared line. All the constraints are listed as matrices in Fig. 6.



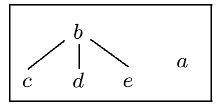
A line drawing

Fig. 5. The left is a line drawing, and the right is a table of the labelings for various junctions. The letter above each labeling of a junction is its name by which the labeling is referred to in the rest of this section.

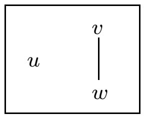
A distinctive feature of this model is that the values of a variable have complex structures and there is some natural relationship among them. Consider the values for a Fork junction in Fig. 5. Values c, d, and e have an edge labeled as $-$, and all three edges of b are labeled as $-$. We can let b be the parent of c, d and e, resulting in the subtree $\{b, c, d, e\}$ in Fig. 6(a). Since value a has nothing to do with the rest, it forms a tree itself. Similarly, we have the forests for Arrow values in Fig. 6(b) and Ell values in Fig. 6(c). Under these forests, the constraints are locally chain convex and strongly union closed. For example, consider the constraint c_{21} on variables x_2 and x_1 in Fig. 6. The domain of x_1 is shown in Fig. 6 (a), and that of x_2 in Fig. 6(b). It can be verified that the image of every subchain of the forest of x_2 is a subchain of the forest of x_1. Note that an empty set is taken as a (trivial) subchain of any tree.

By Theorem 1, this network is globally consistent after arc and path consistency are enforced on it. In this example, we have identified the forest structures for the domains in an intuitive and meaningful way. A more general lesson is that by studying the semantics of domain values, we could discover more efficient constraint solving techniques.

$$c_{21} = \begin{array}{c} abcde \\ \begin{bmatrix} 01100 \\ 10000 \\ 10000 \end{bmatrix} \end{array} \begin{array}{c} u \\ v \\ w \end{array} \quad c_{31} = \begin{bmatrix} 01010 \\ 10000 \\ 10000 \end{bmatrix} \quad c_{51} = \begin{bmatrix} 01001 \\ 10000 \\ 10000 \end{bmatrix}$$



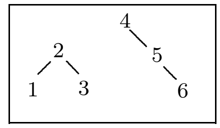
(a) Fork



(b) Arrow

$$c_{24} = c_{37} = c_{56} = \begin{array}{c} 123456 \\ \begin{bmatrix} 001000 \\ 000001 \\ 000110 \end{bmatrix} \end{array} \begin{array}{c} u \\ v \\ w \end{array}$$

$$c_{26} = c_{34} = c_{57} = \begin{bmatrix} 100000 \\ 000100 \\ 000011 \end{bmatrix}$$



(c) Ell

Fig. 6. The constraints for labeling the drawing in Fig. 5

6 Related Work and Conclusion

Jeavons and colleagues have done a series of work to characterize the complexity of constraint languages [3]. A constraint language is parameterized by a set. Given a set D, a *constraint language* over D is a set of relations with finite arity. Given a language L over D, the constraint satisfaction problems associated with L, denoted by CSP(L), are a triple (V, D, C) where V is an arbitrary set of variables, D (over which L is defined) the domain of each variable of V, and C the set of constraints over the variables such that each $c \in C$ belongs to L. A constraint language L over D is *tractable* if CSP(L') can be solved in polynomial time, for each finite subset $L' \subset L$. Several types of *polymorphism* have been identified to characterize the tractable languages. In this paper, instead of a constraint language, we consider the tractability of a set of problems $(V, \mathcal{D}, C)$ where $V = \{1, 2, \ldots, n\}$, $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ and D_i (an arbitrary finite set) is the domain of variable i ($i \in 1..n$), and C a set of constraints. Our result shows that this set of problems can be solved in polynomial time when certain convexity properties are satisfied.

Although the tractability of a constraint language seems to be the same as the tractability of a set of problems, they are indeed different. The key difference lies in that a constraint language involves a fixed domain D and a fixed set of relations L. All variables in different instances of CSP(L) have to have the same domain D. A recent work [2] has generalized constraint languages to *multi-sorted* constraint languages that are over more than one set. For a multi-sorted constraint language L over $\{D_1, D_2, \ldots, D_k\}$, the variables in CSP($L$) are allowed to take any D_i ($i \in 1..k$) as their domains. It is shown that even this simple extension has serious consequences for the characterization of constraint languages: "... [the original constraint languages] can in fact mask the difference between tractability and NP-completeness for some languages, ..." [2]. Not all results for constraint languages hold for multi-sorted languages. There is still a gap between a multi-sorted language and a set of problems. In the CSPs associated with a multi-sorted language, the domains of variables are restricted to a *fixed* collection of sets while in a set of problems, arbitrary set is allowed to be the domain of a variable. The knowledge is still absent on how algebraic operations can be used to directly characterize the tractability of a set of problems.

To have a better understanding of the relationship between our result and the results on constraint languages, we focus on constraint languages. Consider a constraint language L over D. Particularly, every relation $R \in L$ satisfies the convexity property mentioned in Theorem 1. Since enforcing arc and path consistency guarantees global consistency of CSP(L) (by Theorem 1), L must have a near-unanimity polymorphism by the result in [8]. In this situation, the result in [8] gives a general characterization ("indirectly" through algebraic operations) of all constraint languages on which enforcing local (k-) consistency ensures global consistency while our result helps to identify a "concrete" subclass of these languages ("directly" through the convexity properties of the constraints).

Based on the work reported here, Kumar [10] has proposed a more general property on tree convexity – *arc consistent consecutive tree convexity (ACCTC)* – such that the problems with that property are tractable. Radically different from our and Jeavons and colleagues' approaches, Kumar uses random algorithms as a tool to show the tractability of the problems of concern. Due to the nature of our approach, enforcing arc and path consistency on our proposed class of problems ensures global consistency, and there is efficient deterministic algorithms to achieve the global consistency [12,1]. For the ACCTC problems, it is not known whether there is efficient deterministic algorithms, neither is it known whether the arc and path consistency ensures global consistency. The tree convexity of a constraint network can be recognized efficiently [17]. Kumar also observes that an algorithm in [4] can be used to recognize the tree convexity of a network although no algorithm is presented to recognize the ACCTC of a network. As the case of connected row convexity and ACCTC, how to recognize efficiently whether a constraint network is locally chain convex and strictly union closed is an open problem.

We have presented some properties of tree convex constraints that are closed under intersection and/or composition. As a result, we identified a new tractable class of networks – locally chain convex and strictly union closed networks – on which enforcing arc and path consistency on them ensures global consistency. This result generalizes the existing work on convexity of constraints, e.g., [5], and reveals a more fundamental property – local chain convexity and strict union closedness – that determines the tractability of a class of convex constraints. Our result also shows a direct interaction between the semantics of constraints and the semantics of domain values in deciding a tractable class of problems. This interaction is reflected in the properties of intersection and composition of tree convex constraints. An application of the new tractable class of networks is also presented, demonstrating that tree convexity is a useful and natural way to characterize the semantics of domain values, in addition to the traditional ones like total ordering.

References

[1] C. Bessiere, J.C. Regin, R.H.C. Yap, and Y. Zhang. An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, 165(2):165–185, 2005.

[2] Andrei A. Bulatov and Peter Jeavons. An algebraic approach to multi-sorted constraints. In *CP*, pages 183–198, 2003.

[3] David Cohen and Peter Jeavons. The complexity of constraint languages. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, pages 245–280. Elsevier, 2006.

[4] Vincent Conitzer, Jonathan Derryberry, and Tuomas Sandholm. Combinatorial auctions with structured item graphs. In *AAAI*, pages 212–218, 2004.

[5] Y. Deville, O. Barette, and P. Van Hentenryck. Constraint satisfaction over connected row convex constraints. In *Proceedings of International Joint Conference on Artificial Intelligence 1997*, volume 1, pages 405–411, Nagoya, Japan, 1997. IJCAI Inc. (See also Artificial Intelligence 109(1999): 243–271).

[6] E.C. Freuder. Synthesizing constraint expressions. *Communications of ACM*, 21(11):958–966, 1978.

[7] P. G. Jeavons, D. A. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of The ACM*, 44(4):527–548, 1997.

[8] Peter Jeavons, David A. Cohen, and Martin C. Cooper. Constraints, consistency and closure. *Artif. Intell.*, 101(1-2):251–265, 1998.

[9] L. M. Kirousis and C. H. Papadimitriou. The complexity of recognizing polyhedral scenes. In *Journal of Computer and System Sciences*, volume 37, pages 14–38, 1988.

[10] T. K. Satish Kumar. Simple randomized algorithms for tractable row and tree convex constraints. In *AAAI*, 2006.

[11] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):118–126, 1977.

[12] R. Mohr and T.C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[13] U. Montanari. Networks of constraints: fundamental properties and applications. *Information Science*, 7(2):95–132, 1974.

[14] P. van Beek and R. Dechter. On the minimality and global consistency of row-convex constraint networks. *Journal of The ACM*, 42(3):543–561, 1995.

[15] D. L. Waltz. Generating semantic descriptions from drawings of scenes with shadows. Technical Report MAC-AI-TR-271, MIT, Cambridge, MA, 1972.

[16] Patrick Henry Winston. *Artificial Intelligence.* Addison-Wesley, Reading, Massachusetts, third edition, 1992.

[17] Guy Yosiphon. Efficient algorithm for identifying tree convex constraints. Manuscript, 2003.

[18] Yuanlin Zhang and Eugene C. Freuder. Tractable tree convex constraints. In *Proceedings of National Conference on Artificial Intelligence 2004*, pages 197–202, San Jose, CA, USA, 2004. AAAI press.

[19] Yuanlin Zhang and Roland H. C. Yap. Making AC-3 an optimal algorithm. In *Proceedings of International Joint Conference on Artificial Intelligence 2001*, pages 316–321, Seattle, 2001. IJCAI Inc.

[20] Yuanlin Zhang and Roland H. C. Yap. Consistency and set intersection. In *Proceedings of International Joint Conference on Artificial Intelligence 2003*, pages 263–268, Acapulco, Mexico, 2003. IJCAI Inc.