On Tree-Preserving Constraints

Shufeng Kong[1], Sanjiang Li[1]*, Yongming Li[2], and Zhiguo Long[1]

[1] QCIS, FEIT, University of Technology Sydney, Australia
Shufeng.Kong@student.uts.edu.au
Sanjiang.Li@uts.edu.au
Zhiguo.Long@student.uts.edu.au
[2] College of Computer Science, Shaanxi Normal University, China
liyongm@snnu.edu.cn

Abstract. Tree convex constraints are extensions of the well-known row convex constraints. Just like the latter, every path-consistent tree convex constraint network is globally consistent. This paper studies and compares three subclasses of tree convex constraints which are called chain-, path- and tree-preserving constraints respectively. While the tractability of the subclass of chain-preserving constraints has been established before, this paper shows that every chain- or path-preserving constraint network is in essence the disjoint union of several independent connected row convex constraint networks, and hence (re-)establish the tractability of these two subclasses of tree convex constraints. We further prove that, when enforcing arc- and path-consistency on a tree-preserving constraint network, in each step, the network remains tree-preserving. This ensures the global consistency of the tree-preserving network if no inconsistency is detected. Moreover, it also guarantees the applicability of the partial path-consistency algorithm to tree-preserving constraint networks, which is usually more efficient than the path-consistency algorithm for large sparse networks. As an application, we show that the class of tree-preserving constraints is useful in solving the scene labelling problem.

1 Introduction

Constraint satisfaction problems (CSPs) have been widely used in many areas, such as scene labeling [10], natural language parsing [15], picture processing [16], and spatial and temporal reasoning [5,14]. Since deciding consistency of CSP instances is NP-hard in general, lots of efforts have been devoted to identify tractable subclasses. These subclasses are usually obtained by either restricting the topology of the underlying graph of the constraint network (being a tree or having treewidth bounded by a constant) or restricting the type of the allowed constraints between variables (cf. [17]).

In this paper, we are mainly interested in the second type of restriction. Montanari [16] shows that path-consistency is sufficient to guarantee that a network is globally consistent if the relations are all monotone. Van Beek and Dechter [17] generalise monotone constraints to a larger class of row convex constraints, which are further generalised to tree convex constraints by Zhang and Yap [20]. These constraints also have the nice property that every path-consistent constraint network is globally consistent.

* Corresponding author.

However, neither row convex constraints nor tree convex constraints are closed under composition and intersection, the main operations of path-consistent algorithms. This means enforcing path-consistency may destroy row and tree-convexity. Deville et al. [6] propose a tractable subclass of row convex constraints, called connected row convex (CRC) constraints, which are closed under composition and intersection. Zhang and Freuder [18] also identify a tractable subclass for tree convex constraints, called *locally chain convex and strictly union closed* constraints. They also propose the important notion of consecutive constraints. Kumar [13] shows that the subclass of arc-consistent consecutive tree convex (ACCTC) constraints is tractable by providing a polynomial time randomised algorithm. But, for the ACCTC problems, "it is not known whether there are efficient deterministic algorithms, neither is it known whether arc- and path-consistency ensures global consistency on those problems." [18]

In this paper, we study and compare three subclasses of tree convex constraints which are called, respectively, chain-, path- and tree-preserving constraints. Chain-preserving constraints are exactly "locally chain convex and strictly union closed" constraints and ACCTC constraints are strictly contained in the subclass of tree-preserving constraints. We first show that every chain- or path-preserving constraint network is in essence the disjoint union of several independent CRC constraint networks and then prove that enforcing arc- and path-consistency on a tree-preserving constraint network ensures global consistency. This provides an affirmative answer to the above open problem raised in [18]. Note also that our result is more general than that of Kumar [13] as we do not require the constraint network to be arc-consistent. Moreover, when enforcing arc- and path-consistent on a tree-preserving constraint network, in each step, the network remains tree-preserving. This guarantees the applicability of the partial path-consistency algorithm [2] to tree-preserving constraint networks, which is usually more efficient than the path-consistency algorithm for large sparse networks. We further show that a large subclass of the trihedral scene labelling problem [10,12] can be modelled by tree-preserving constraints.

In the next section, we introduce basic notations and concepts that will be used throughout the paper. Chain-, path-, and tree-preserving constraints are discussed in Sections 3, 4, and 5, respectively. Application of tree-preserving constraints in the scene labelling problem is shown in Section 6. Section 7 briefly discusses the connection with majority operators [11] and concludes the paper.

2 Preliminaries

Let D be a domain of a variable x. A graph structure can often be associated to D such that there is a bijection between the vertices in the graph and the values in D. If the graph is connected and acyclic, i.e. a tree, then we say it is a *tree domain* of x. Tree domains arise naturally in e.g. scene labeling [18] and combinatorial auctions [4]. We note that, in this paper, we have a specific tree domain D_x for each variable x.

In this paper, we distinguish between tree and rooted tree. Standard notions from graph theory are assumed. In particular, the *degree* of a node a in a graph G, denoted by $\deg(a)$, is the number of neighbors of a in G.

Definition 1. *A* tree *is a connected graph without any cycle. A tree is* rooted *if it has a specified node r, called the* root *of the tree. Given a tree T, a subgraph I is called a* subtree *of T if I is connected. An empty set is a subtree of any tree.*

Let T be a (rooted) tree and I a subtree of T. I is a path *(*chain*, resp.) in T if each node in I has at most two neighbors (at most one child, resp.) in I. Given two nodes p, q in T, the unique path that connects p to q is denoted by $\pi_{p,q}$.*

Suppose a is a node of a tree T. A branch *of a is a connected component of $T \setminus \{a\}$.*

Throughout this paper, we always associate a subtree with its node set.

Definition 2. *A binary constraint has the form $(x\delta y)$, where x, y are two variables with domains D_x and D_y and δ is a binary relation from D_x to D_y, or $\delta \subseteq D_x \times D_y$. For simplicity, we often denote by δ this constraint. A value $u \in D_x$ is* supported *if there exists a value v in D_y s.t. $(u, v) \in \delta$. In this case, we say v is a* support *of u. We say a subset F of D_x is* unsupported *if every value in F is not supported. Given $A \subseteq D_x$, the* image *of A under δ is defined as $\delta(A) = \{b \in D_y : (\exists a \in A)(a, b) \in \delta\}$. For $A = \{a\}$ that contains only one value, without confusion, we also use $\delta(a)$ to represent $\delta(\{a\})$.*

A binary constraint network consists of a set of variables $V = \{x_1, x_2, ..., x_n\}$ with a finite domain D_i for each variable $x_i \in V$, and a set Δ of binary constraints over the variables of V. The usual operations on relations, e.g., intersection ($\cap$), composition ($\circ$), and inverse ($^{-1}$), are applicable to constraints. As usual, we assume that there is at most one constraint for any ordered pair of variables (x, y). Write δ_{xy} for this constraint if it exists. In this paper, we always assume δ_{xy} is the inverse of δ_{yx}, and if there is no constraint for (x, y), we assume δ_{xy} is the universal constraint.

Definition 3. *[8,9] A constraint network Δ over n variables is k-*consistent *iff any consistent instantiation of any distinct $k-1$ variables can be consistently extended to any k-th variable. We say Δ is* strongly k-consistent *iff it is j-consistent for all $j \leq k$; and say Δ is* globally consistent *if it is strongly n-consistent. 2- and 3-consistency are usually called* arc- *and* path-consistency *respectively.*

Definition 4. *Let x, y be two variables with finite tree domains $T_x = (D_x, E_x)$ and $T_y = (D_y, E_y)$ and δ a constraint from x to y. We say δ, w.r.t. T_x and T_y, is*

- tree convex *if the image of every value a in D_x (i.e. $\delta(a)$) is a (possibly empty) subtree of T_y;*
- consecutive *if the image of every edge in T_x is a subtree in T_y;*
- path-preserving *if the image of every path in T_x is a path in T_y.*
- tree-preserving *if the image of every subtree in T_x is a subtree in T_y.*

In case T_x and T_y are rooted, we say δ, w.r.t. T_x and T_y, is

- chain-preserving *if the image of every chain in T_x is a chain in T_y.*

Chain-preserving constraints are exactly those "locally chain convex and strictly union closed" constraints defined in [18].

CRC constraints are special tree convex constraints defined over chain domains. The following definition of CRC constraints is equivalent to the one given in [6].

Definition 5. *Let x, y be two variables with finite tree domains T_x and T_y, where T_x and T_y are chains. A constraint δ from x to y is* connected row convex *(CRC), w.r.t. T_x and T_y, if both δ and δ^{-1} are chain-preserving.*

The class of CRC constraints is tractable and closed under intersection, inverse, and composition [6].

Definition 6. *A binary constraint network Δ over variables in V and tree domains T_x $(x \in V)$ is called tree convex, chain-, path-, or tree-preserving if every constraint $\delta \in \Delta$ is tree convex, chain-, path-, or tree-preserving, respectively. A CRC constraint network is defined similarly.*

Proposition 1. *Every chain-, path-, or tree-preserving constraint (network) is consecutive and every path-preserving constraint (network) is tree-preserving. Moreover, every arc-consistent consecutive tree convex (ACCTC) constraint (network) is tree-preserving.*

Not every consecutive tree convex constraint (or chain-preserving constraint) is tree-preserving, but such a constraint becomes tree-preserving if it is arc-consistent.

Lemma 1. *[20] Let T be a tree and suppose t_i $(i = 1, .., n)$ are subtrees of T. Then $\bigcap_{i=1}^{n} t_i$ is nonempty iff $t_i \cap t_j$ is nonempty for every $1 \leq i \neq j \leq n$.*

Lemma 2. *Let T be a tree and t, t' subtrees of T. Suppose $\{u, v\}$ is an edge in T. If $u \in t$ and $v \in t'$, then $t \cup t'$ is a subtree of T; if, in addition, $u \notin t'$ and $v \notin t$, then $t \cap t' = \varnothing$.*

Using Lemma 1, Zhang and Yap [20] proved

Theorem 1. *A tree-convex constraint network is globally consistent if it is path-consistent.*

3 Chain-Preserving Constraints

Zhang and Freuder [18] have proved that any consistent chain-preserving network can be transformed to an equivalent globally consistent network by enforcing arc- and path-consistency. This implies that the class of chain-preserving constraints is tractable. We next show that every chain-preserving constraint network Δ can be uniquely divided into a small set of k CRC constraint networks $\Delta_1, ..., \Delta_k$ s.t. Δ is consistent iff at least one of Δ_i is consistent.

We first recall the following result used in the proof of [18, Theorem 1].

Proposition 2. *Let Δ be a chain-preserving constraint network over tree domains T_x $(x \in V)$. If no inconsistency is detected, then Δ remains chain-preserving after enforcing arc-consistency.*

We note that these tree domains over variables in V may need adjustment in the process of enforcing arc-consistency. Here by *adjustment* we mean adding edges to the tree structure so that it remains connected when unsupported values are deleted.

Definition 7. *Let T be a tree with root. A chain $[a, a^*]$ in T is called an* irreducible perfect chain *(ip-chain) if (i) a is the root or has one or more siblings; (ii) a^* is a leaf node or has two or more children; and (iii) every node in $[a, a^*)$ has only one child.*

Note that it is possible that $a = a^*$. In fact, this happens when a is the root or has one or more siblings and has two or more children. An ip-chain as defined above is a minimum chain which satisfies (1) in the following lemma.

Lemma 3. *Suppose δ_{xy} and δ_{yx} are arc-consistent and chain-preserving w.r.t. rooted trees T_x and T_y. Assume $[a, a^*] \subseteq T_x$ is an ip-chain. Then*

$$\delta_{yx}(\delta_{xy}([a, a^*])) = [a, a^*] \tag{1}$$

and $\delta_{xy}([a, a^*])$ is also an ip-chain in T_y.

Proof. W.l.o.g., we suppose $\hat{a}$ is the parent of a, a' is a sibling of a, and $a_1, a_2, ..., a_k$ ($k \geq 2$) are the children of a^*.

Because δ_{xy} and δ_{yx} are arc-consistent and chain-preserving, $\delta_{xy}([a, a^*])$ is a non-empty chain in T_y, written $[b, b^*]$, and so is $\delta_{yx}([b, b^*])$. Suppose $\delta_{yx}([b, b^*])$ is not $[a, a^*]$. This implies that either $\hat{a}$ or one of $a_1, a_2, ..., a_k$ is in $\delta_{yx}([b, b^*])$.

Suppose $\hat{a} \in \delta_{yx}([b, b^*])$. Then there exists $\hat{b} \in [b, b^*]$ such that $(\hat{a}, \hat{b}) \in \delta_{xy}$. By $\hat{b} \in [b, b^*] = \delta_{xy}([a, a^*])$, we have $a^+ \in [a, a^*]$ s.t. $(a^+, \hat{b}) \in \delta_{xy}$. Therefore, $[\hat{a}, a^+]$ is contained in $\delta_{yx}(\delta_{xy}(\{\hat{a}\}))$. Recall that a' is a sibling of a. Because $\delta_{yx}(\delta_{xy}([\hat{a}, a']))$ contains $\hat{a}, a', a^+$, it cannot be a chain in T_x. A contradiction. Therefore, $\hat{a} \notin \delta_{yx}([b, b^*])$.

Suppose, for example, $a_1 \in \delta_{yx}([b, b^*])$. Then there exist $b' \in [b, b^*]$ s.t. $(a_1, b') \in \delta_{xy}$ and $\bar{a} \in [a, a^*]$ s.t. $(\bar{a}, b') \in \delta_{xy}$. We have $\delta_{yx}(\delta_{xy}(\{\bar{a}\}) \supseteq [\bar{a}, a_1]$ and $\delta_{yx}(\delta_{xy}([\bar{a}, a_2])$ contains $\{\bar{a}, a_1, a_2\}$, which is not a subset of a chain. Therefore, $a_i \notin \delta_{yx}([b, b^*])$.

So far, we have proved $\delta_{yx}(\delta_{xy}([a, a^*])) = [a, a^*]$. We next show $[b, b^*]$ is also an ip-chain. First, we show every node in $[b, b^*)$ has only one child. Suppose not and $b' \in [b, b^*)$ has children b_1, b_2 with $b_1 \in (b', b^*]$. Since $\delta_{xy}([\hat{a}, a^*])$ is a chain that contains $[b, b^*]$, we know $(\hat{a}, b_2)$ is not in δ_{xy}. Furthermore, as $\delta_{yx}(\{b', b_2\})$ is a chain in T_x and the image of b_2 is disjoint from $[a, a^*]$, we must have $(a_i, b_2) \in \delta_{xy}$ for some child a_i of a^*. Note that then $\delta_{xy}([a, a_i])$ contains $[b, b^*]$ and b_2 and thus is not a chain. This contradicts the chain-preserving property of δ_{xy}. Hence, every node in $[b, b^*)$ has only one child. In other words, $[b, b^*]$ is contained in an ip-chain $[u, v]$.

By the result we have proved so far, we know $\delta_{xy}(\delta_{yx}([u, v])) = [u, v]$ and $\delta_{yx}([u, v])$ is contained in an ip-chain in T_x. Because $[a, a^*] = \delta_{yx}([b, b^*]) \subseteq \delta_{yx}([u, v])$ is an ip-chain, we know $\delta_{yx}([u, v])$ is exactly $[a, a^*]$. Therefore, we have $[u, v] = \delta_{xy}(\delta_{yx}([u, v]))$ $= \delta_{xy}([a, a^*]) = [b, b^*]$. This proves that $[b, b^*]$ is an ip-chain in T_y. □

Using the above result, we can break T_x into a set of ip-chains by deleting the edges from each node a to its children if a has two or more children. Write $\mathcal{I}_x$ for the set of ip-chains of T_x. Similar operation and notation apply to T_y. It is clear that two different ip-chains in $\mathcal{I}_x$ are disjoint and δ_{xy} naturally gives rise to a bijection from $\mathcal{I}_x$ to $\mathcal{I}_y$.

Lemma 4. *Suppose Δ is an arc-consistent and chain-preserving constraint network over tree domains T_x ($x \in V$). Fix a variable $x \in V$ and let $\mathcal{I}_x = \{I_x^1, ..., I_x^l\}$ be*

the set of ip-chains of T_x. Then, for every $y \neq x$ in V, the set of ip-chains in T_y is $\{\delta_{xy}(I_x^1), \delta_{xy}(I_x^2), ..., \delta_{xy}(I_x^l)\}$. Write Δ_i for the restriction of Δ to I_x^i. Then each Δ_i is a CRC constraint network and Δ is consistent iff at least one Δ_i is consistent.

The following result asserts that the class of chain-preserving constraints is tractable.

Theorem 2. *Let Δ be a chain-preserving constraint network. If no inconsistency is detected, then enforcing arc- and path-consistency determines the consistency of Δ and transforms Δ into a globally consistent network.*

Proof. First, by Proposition 2, we transform Δ into an arc-consistent and chain-preserving constraint network if no inconsistency is detected. Second, by Lemma 4, we reduce the consistency of Δ to the consistency of the CRC constraint networks $\Delta_1, ..., \Delta_l$. By [6], we know enforcing path-consistency transforms a CRC constraint network into a globally consistent one if no inconsistency is detected. If enforcing arc- and path-consistency does not detect any inconsistency, then the result is a set of at most l globally consistent CRC networks Δ_i', the union of which is globally consistent and equivalent to Δ. □

Lemma 4 also suggests that we can use the variable elimination algorithm for CRC constraints [19] to more efficiently solve chain-preserving constraints.

4 Path-Preserving Constraints

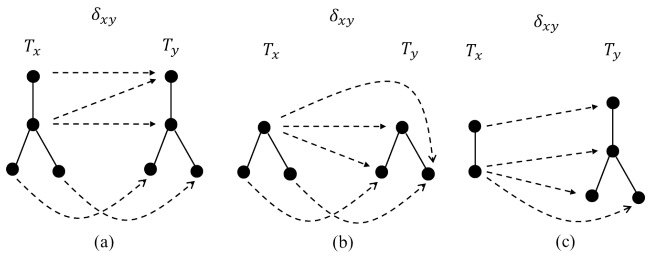


Fig. 1. (a) A chain- but not path-preserving constraint; (b) A path- but not chain-preserving constraint; (c) A tree-preserving but neither path- nor chain-preserving constraint.

At first glance, path-preserving constraints seem to be more general than chain-preserving constraints, but Fig. 1(a,b) show that they are in fact incomparable.

We show the class of path-preserving constraints is also tractable by establishing its connection with CRC constraints.

We have the following simple results.

Lemma 5. *Suppose δ_{xy} and δ_{yx} are path-preserving (tree-preserving) w.r.t. tree domains T_x and T_y. Let t be a subtree of T_x and δ'_{xy} and δ'_{yx} the restrictions of δ_{xy} and δ_{yx} to t. Then both δ'_{xy} and δ'_{yx} are path-preserving (tree-preserving).*

Lemma 6. *Suppose δ_{xy} is nonempty and path-preserving (tree-preserving) w.r.t. tree domains T_x and T_y. If $v \in T_y$ has no support in T_x under δ_{yx}, then all supported nodes of T_y are in the same branch of v. That is, every node in any other branch of v is not supported under δ_{yx}.*

Proof. Suppose a, b are two supported nodes in T_y. There exist u_1, u_2 in T_x s.t. $u_1 \in \delta_{yx}(a)$ and $u_2 \in \delta_{yx}(b)$. By $\delta_{yx} = \delta_{xy}^{-1}$, we have $a \in \delta_{xy}(u_1)$ and $b \in \delta_{xy}(u_2)$. Hence $a, b \in \delta_{xy}(\pi_{u_1,u_2})$. Since δ_{xy} is path-preserving (tree-preserving), $\delta_{xy}(\pi_{u_1,u_2})$ is a path (tree) in T_y. If a, b are in two different branches of v, then $\pi_{a,b}$ must pass v and hence we must have $v \in \delta_{xy}(\pi_{u_1,u_2})$. This is impossible as v has no support. □

It is worth noting that this lemma does not require δ_{yx} to be path- or tree-preserving.

The following result then follows directly.

Proposition 3. *Let Δ be a path-preserving (tree-preserving) constraint network over tree domains T_x ($x \in V$). If no inconsistency is detected, then Δ remains path-preserving (tree-preserving) after enforcing arc-consistency.*

Proof. Enforcing arc-consistency on Δ only removes values which have no support under some constraints. For any $y \in V$, if v is an unsupported value in T_y, then, by Lemma 6, every supported value of T_y is located in the same branch of v. Deleting all these unsupported values from T_y, we get a subtree t of T_y. Applying Lemma 5, the restricted constraint network to t remains path-preserving (tree-preserving). □

Definition 8. *Let T be a tree. A path π in T is* maximal *if there exists no path π' in T that strictly contains π.*

We need three additional lemmas to prove the main result.

Lemma 7. *Suppose δ_{xy} and δ_{yx} are arc-consistent and path-preserving w.r.t. tree domains T_x and T_y. If π is a maximal path in T_x, then $\delta_{xy}(\pi)$ is a maximal path in T_y.*

Lemma 8. *Suppose δ_{xy} and δ_{yx} are arc-consistent and path-preserving w.r.t. T_x and T_y. Assume a is a node in T_x with $\deg(a) > 2$. Then there exists a unique node $b \in T_y$ s.t. $(a, b) \in \delta_{xy}$. Moreover, $\deg(a) = \deg(b)$.*

Proof. Suppose $\pi = a_0 a_1 ... a_k$ is a maximal path in T_x and $\pi^* = b_0 b_1 ... b_l$ is its image under δ_{xy} in T_y. W.l.o.g. we assume $k, l \geq 1$. Suppose a_i is a node in π s.t. $\deg(a_i) > 2$ and $a' \notin \pi$ is another node in T_x s.t. $\{a_i, a'\}$ is an edge in T_x. Suppose $\delta_{xy}(a_i) = [b_j, b_{j'}]$ and $j' > j$.

Because π is a maximal path and π^* is its image, we know $\delta_{xy}(a') \cap \pi^* = \varnothing$. Consider the edge $\{a', a_i\}$. Since $\delta_{xy}(\{a', a_i\})$ is a path in T_y, there exists a node $b' \in \delta_{xy}(a')$ s.t. either $\{b', b_j\}$ or $\{b', b_{j'}\}$ is an edge in T_y. Suppose w.l.o.g. $\{b', b_{j'}\}$ is in T_y. Note that $\pi^* = [b_0, b_l]$ is contained in the union of $\delta_{xy}([a', a_0])$ and $\delta_{xy}([a', a_k])$. In particular, b_l is in either $\delta_{xy}([a', a_0])$ or $\delta_{xy}([a', a_k])$. Let us assume $b_l \in \delta_{xy}([a', a_0])$. Then $b_l, b_j, b_{j'}, b'$ (which are not on any path) are contained in the path $\delta_{xy}([a', a_0])$, a contradiction. Therefore, our assumption that $\delta_{xy}(a_i) = [b_j, b_{j'}]$ and $j' > j$ is incorrect. That is, the image of a_i under δ_{xy} is a singleton, say, $\{b_j\}$. We next show $\deg(b_j) = \deg(a_i)$.

Because $\delta_{xy}(a_i) = \{b_j\}$, the image of each neighbor of a_i in T_x must contain a neighbor of b_j, as δ_{xy} is path-preserving. Moreover, two different neighbors a'_i, a''_i of a_i cannot map to the same neighbor b'_j of b_j. This is because the image of b'_j under δ_{yx}, which is a path in T_x, contains a'_i and a''_i, and hence also contains a_i. This contradicts the assumption $\delta_{xy}(a_i) = \{b_j\}$. This shows that $\deg(a_i) = \deg(b_j)$. □

Definition 9. *Let T be a tree. A path π from a to a^* in T is called an* irreducible perfect path *(ip-path) if (i) every node on path π has degree 1 or 2; and (ii) any neighbour of a (or a^*) that is not on π has degree 3 or more.*

Let $F_x = \{a \in T_x : \deg(a) > 2\}$ and $F_y = \{b \in T_y : \deg(b) > 2\}$. Then δ_{xy}, when restricted to F_x, is a bijection from F_x to F_y. Removing all edges incident to a node in F_x, we obtain a set of pairwise disjoint paths in T_x. These paths are precisely the ip-paths of T_x. Write $\mathcal{P}_x$ for this set. Then δ_{xy} induces a bijection from $\mathcal{P}_x$ to $\mathcal{P}_y$.

Lemma 9. *Suppose Δ is an arc-consistent and path-preserving constraint network over tree domains T_x ($x \in V$). Fix a variable $x \in V$ and let $\mathcal{P}_x = \{\pi_x^1, ..., \pi_x^l\}$ be the set of ip-paths in T_x. Then, for every $y \neq x$, the set of ip-paths in T_y is $\{\delta_{xy}(\pi_x^1), ..., \delta_{xy}(\pi_x^l)\}$. Write Δ_i for the restriction of Δ to π_x^i. Then each Δ_i is a CRC constraint network and Δ is consistent iff at least one Δ_i is consistent.*

Thus the class of path-preserving constraints is tractable.

Theorem 3. *Let Δ be a path-preserving constraint network. If no inconsistency is detected, then enforcing arc- and path-consistency determines the consistency of Δ and transforms Δ into a globally consistent network.*

The proof is analogous to that of Theorem 2. Lemma 9 suggests that we can use the variable elimination algorithm for CRC constraints [19] to more efficiently solve path-preserving constraints.

5 Tree-Preserving Constraints

It is easy to see that every arc-consistent chain- or path-preserving constraint is tree-preserving, but Fig. 1(c) shows that the other direction is not always true.

In this section, we show that the class of tree-preserving constraints is tractable. Given a tree-preserving constraint network Δ, we show that, when enforcing arc- and path-consistency on Δ, in each step, the network remains tree-preserving. Hence, enforcing arc- and path-consistency on Δ will transform it to an equivalent globally consistent network if no inconsistency is detected. Moreover, we show that the partial path-consistency algorithm (PPC) of [2] is applicable to tree-preserving constraint networks. PPC is more efficient than path-consistency algorithm for large sparse constraints.

5.1 Enforcing Arc- and Path-Consistency Preserves Tree-Preserving

Unlike CRC and chain-preserving constraints, removing a value from a domain may change the tree-preserving property of a network. Instead, we need to remove a 'trunk' from the tree domain or just keep one branch.

Definition 10. *Suppose $a \neq b$ are two nodes of a tree T that are not neighbors. The* trunk *between a, b, written M_{ab}, is defined as the connected component of $T \setminus \{a, b\}$ which contains all internal nodes of $\pi_{a,b}$ (see Fig.2). The* M-contraction *of T by $M_{a,b}$, denoted by $T \ominus M_{a,b}$, is the tree obtained by removing nodes in $M_{a,b}$ and adding an edge $\{a, b\}$ to T.*

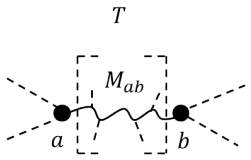


Fig. 2. M_{ab} is a trunk of tree T.

Lemma 10. *Suppose δ_{xy} and δ_{yx} are arc-consistent and tree-preserving w.r.t. tree domains T_x and T_y. Suppose a, b are two nodes in T_x s.t. $\delta_{xy}(a) \cup \delta_{xy}(b)$ is not connected in T_y. Then there exist $r, s \in T_y$ s.t. $r \in \delta_{xy}(a)$, $s \in \delta_{xy}(b)$, and $\delta_{yx}(M_{r,s}) \subseteq M_{a,b}$. Let T_y^* be the domain obtained by deleting from T_y all nodes v s.t. $\delta_{yx}(v) \subseteq M_{a,b}$. Then T_y^* becomes a tree if we add the edge $\{r, s\}$. Moreover, δ_{xy} and δ_{yx} remain arc-consistent and tree-preserving when restricted to $T_x \ominus M_{a,b}$ and T_y^*.*

Proof. Choose $r \in \delta_{xy}(a)$ and $s \in \delta_{xy}(b)$ such that the path $\pi_{r,s}$ from r to s in T_y is a shortest one among $\{\pi_{r',s'} : r' \in \delta_{xy}(a), s' \in \delta_{xy}(b)\}$. In particular, we have $\pi_{r,s} \cap (\delta_{xy}(a) \cup \delta_{xy}(b)) = \{r, s\}$. We assert that the image of every node v in $M_{r,s}$ under δ_{yx} is contained in $M_{a,b}$. Suppose otherwise and there exists u in $T_x \setminus M_{a,b}$ s.t. $(u, v) \in \delta_{xy}$. Assume that u is in the same connected component as a. Since the subtree $\delta_{yx}(\pi_{v,s})$ contains u and b, it also contains a. This implies that there is a node v' on $\pi_{v,s}$ which is in $\delta_{xy}(a)$. This is impossible as $v \in M_{r,s}$ and $\delta_{xy}(a) \cap \pi_{r,s} = \{r\}$. Therefore $\delta_{yx}(v) \subseteq M_{a,b}$ for any $v \in M_{r,s}$. Hence $\delta_{yx}(M_{r,s}) \subseteq M_{a,b}$ holds.

It is clear that, when restricted to $T_x \ominus M_{a,b}$ and $T_y \ominus M_{r,s}$, $\delta_{xy}(\{a, b\})$ is connected and so is $\delta_{yx}(\{r, s\})$. For any other edge $\{a', b'\}$ in $T_x \ominus M_{a,b}$, by $\delta_{yx}(M_{r,s}) \subseteq M_{a,b}$, $\delta_{xy}(\{a', b'\}) \cap M_{r,s} = \varnothing$ and the image of $\{a', b'\}$ is unchanged (hence connected) after the M-contraction of T_y. This shows that δ_{xy} is consecutive when restricted to $T_x \ominus M_{a,b}$. Furthermore, since every node in $T_x \ominus M_{a,b}$ is supported in $T_y \ominus M_{r,s}$, we know δ_{xy} is also tree-preserving when restricted to $T_x \ominus M_{a,b}$.

It is possible that there is a node $v \in T_y \ominus M_{r,s}$ s.t. $\delta_{yx}(v) \subseteq M_{a,b}$. We assert that any v like this has at most one branch in $T_y \setminus M_{r,s}$ s.t. there is a node v' in the branch which is supported under δ_{yx} by a node in $T_x \setminus M_{a,b}$. Because δ_{xy} is tree-preserving when restricted to $T_x \ominus M_{a,b}$, this follows immediately from Lemma 6. This implies that, if we remove all these nodes v s.t. $\delta_{yx}(v) \subseteq M_{a,b}$ from $T_y \ominus M_{r,s}$, the domain is still connected. As a consequence, the two constraints remain arc-consistent and tree preserving when restricted to $T_x \ominus M_{a,b}$ and T_y^*. □

In general, we have

Lemma 11. *Let Δ be an arc-consistent and tree-preserving constraint network over tree domains T_x ($x \in V$). Suppose $x \in V$ and $M_{a,b}$ is a trunk in T_x. When restricted to $T_x \ominus M_{a,b}$ and enforcing arc-consistency, Δ remains tree-preserving if we modify each T_y ($y \in V$) by deleting all unsupported nodes and adding some edges.*

Proof (Sketch). The result follows from Lemmas 10 and 5. One issue we need to take care of is how two trunks interact. Suppose x, y, z are three different variables and M, M' are trunks to be contracted from T_x and T_y respectively. Applying Lemma 10 to the constraints between x and z and, separately, to the constraints between y and z, we get two different trunks, say $M_{a,b}$ and $M_{c,d}$, to be contracted from the same tree domain T_z. Can we do this (i.e. applying Lemma 10) one by one? Does the order of the contractions matter? To answer these questions, we need to know what is exactly the union of two trunks. There are in essence ten configurations as shown in Fig. 3. The union of two trunks can be the whole tree, a branch, a trunk, or two disjoint trunks. If the union is the whole tree, then the network is inconsistent; if it is a branch, then we can remove it directly; if it is a trunk or two disjoint trunks, then we can use Lemma 10 to contract them one by one in either order. □

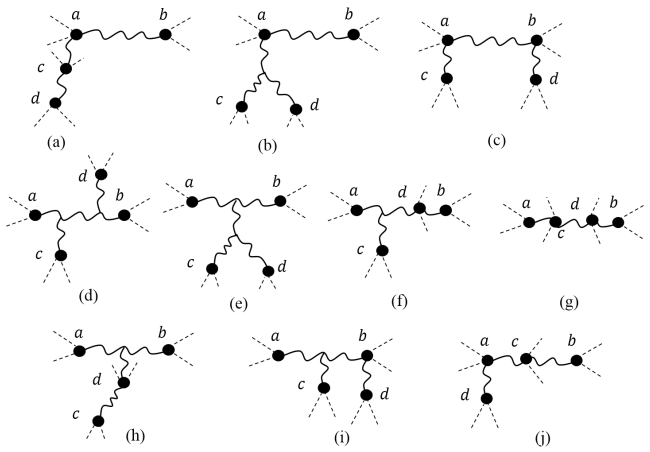


Fig. 3. Possible configurations of trunks $M_{a,b}$ and $M_{c,d}$.

Lemma 12. *Assume δ_{xy} and δ'_{xy} are two arc-consistent and tree-preserving constraints w.r.t. trees T_x and T_y. Let $\delta^*_{xy} = \delta_{xy} \cap \delta'_{xy}$. Suppose $u \in T_x$ and $\delta_{xy}(u) \cap \delta'_{xy}(u) = \varnothing$. Then the supported values of δ^*_{xy} in T_x are in at most two branches of u.*

Proof. Suppose u_1, u_2, u_3 are three supported values of δ^*_{xy} in T_x that are in three different branches of u. Take $w_i \in \delta_{xy}(u_i) \cap \delta'_{xy}(u_i)$. For each i, we have either $w_i \notin \delta_{xy}(u)$ or $w_i \notin \delta'_{xy}(u)$. Recall that π_{w_i,w_j} denotes the unique path π_{w_i,w_j} that connects w_i to w_j ($1 \le i \ne j \le 3$). There are two subcases. (1) One node is on the path that

connects the other two. Suppose w.l.o.g. w_3 is between w_1 and w_2. If $w_3 \notin \delta_{xy}(u)$, then there exist $v_1 \in \pi_{w_1,w_3}$ and $v_2 \in \pi_{w_3,w_2}$ s.t. $v_1, v_2 \in \delta_{xy}(u)$. This is because the image of π_{w_1,w_3} and the image of π_{w_3,w_2} (under δ_{yx}) both contain u. That is, $v_1, v_2 \in \delta_{xy}(u)$ but $w_3 \notin \delta_{xy}(u)$. Since $\delta_{xy}(u)$ is a subtree and w_3 is on the path π_{v_1,v_2}, this is impossible. The case when $w_3 \notin \delta'_{xy}(u)$ is analogous. (2) The three nodes are in three different branches of a node w. In this case, we note there exists a node between any path π_{w_i,w_j} which is a support of u under δ_{xy}. It is easy to see that w itself is a support of u under δ_{xy}. Similarly, we can show w is also a support of u under δ'_{xy}. Since both subcases lead to a contradiction, we know the supported values of δ^*_{xy} are in at most two branches of u. □

Actually, this lemma shows

Corollary 1. *Assume δ_{xy} and δ'_{xy} are two arc-consistent and tree-preserving constraints w.r.t. trees T_x and T_y. Then those unsupported values of $\delta_{xy} \cap \delta'_{xy}$ in T_x are in a unique set of pairwise disjoint branches and trunks.*

Similar to Lemma 10, we have

Lemma 13. *Suppose δ_{xy} and δ'_{xy} are arc-consistent and tree-preserving constraints w.r.t. trees T_x and T_y and so are δ_{yx} and δ'_{yx}. Let $\delta^*_{xy} = \delta_{xy} \cap \delta'_{xy}$. Assume $\{u, v\}$ is an edge in T_x s.t. $\delta^*_{xy}(u) \cup \delta^*_{xy}(v)$ is disconnected in T_y. Then there exist $r \in \delta^*_{xy}(u)$ and $s \in \delta^*_{xy}(v)$ s.t. every node in $M_{r,s}$ is unsupported under δ^*_{yx}.*

Proof. Write $T_r = \delta^*_{xy}(u)$ and $T_s = \delta^*_{xy}(v)$. Clearly, T_r and T_s are nonempty subtrees of T_y. Since they are disconnected, there exist $r \in T_r$, $s \in T_s$ s.t. $\pi_{r,s} \cap (T_r \cup T_s) = \{r, s\}$ (see Fig. 4 for an illustration). Write $A = \delta_{xy}(u)$, $B = \delta_{xy}(v)$, $C = \delta'_{xy}(u)$ and $D = \delta'_{xy}(v)$. We show every node in $M_{r,s}$ is not supported under δ^*_{yx}.

Suppose w is an arbitrary internal node on $\pi_{r,s}$. We first show w is not supported under δ^*_{yx}. Note $w \in A \cup B$, $w \in C \cup D$, $w \notin A \cap C$, and $w \notin B \cap D$. There are two cases according to whether $w \in A$. If $w \in A$, then we have $w \notin C$, $w \in D$, and $w \notin B$. If $w \notin A$, then we have $w \in B$, $w \notin D$, and $w \in C$. Suppose w.l.o.g. $w \in A$. By $w \in A = \delta_{xy}(u)$, we have $u \in \delta_{yx}(w)$; by $w \notin B = \delta_{xy}(v)$, we have $v \notin \delta_{yx}(w)$. Similarly, we have $u \notin \delta'_{yx}(w)$ and $v \in \delta'_{yx}(w)$. Thus subtree $\delta'_{yx}(w)$ is disjoint from subtree $\delta_{yx}(w)$. This shows $\delta^*_{yx}(w) = \varnothing$ and hence w is not supported under δ^*_{yx}.

Second, suppose w_1 is an arbitrary node in $M_{r,s}$ s.t. w_1 is in a different branch of w to r and s, i.e. $\pi_{w,w_1} \cap (T_r \cup T_s) = \varnothing$. We show w_1 is not supported under δ^*_{yx} either.

Again, we assume $w \in A$. In this case, we have $u \in \delta_{yx}(w) \subseteq \delta_{yx}(\pi_{w,w_1})$ and $v \in \delta'_{yx}(w) \subseteq \delta'_{yx}(\pi_{w,w_1})$. As $\pi_{w,w_1} \cap (T_r \cup T_s) = \varnothing$, we have $\pi_{w,w_1} \cap T_r = \pi_{w,w_1} \cap A \cap C = \varnothing$. As $\pi_{w,w_1} \cap A \neq \varnothing$ and $A \cap C \neq \varnothing$, by Lemma 1, we must have $\pi_{w,w_1} \cap \delta'_{xy}(u) = \varnothing$. This shows $u \notin \delta'_{yx}(\pi_{w,w_1})$. Similarly, we can show $v \notin \delta_{yx}(\pi_{w,w_1})$. Thus subtree $\delta'_{yx}(\pi_{w,w_1})$ is disjoint from subtree $\delta_{yx}(\pi_{w,w_1})$ and, hence, $\delta^*_{yx}(\pi_{w,w_1}) = \varnothing$. This proves that w_1 is not supported under δ^*_{yx} either.

In summary, every node in $M_{r,s}$ is unsupported. □

Proposition 4. *[18] Assume δ_{xz} and δ_{zy} are two tree-preserving constraints w.r.t. trees T_x, T_y, and T_z. Then their composition $\delta_{xz} \circ \delta_{zy}$ is tree-preserving.*

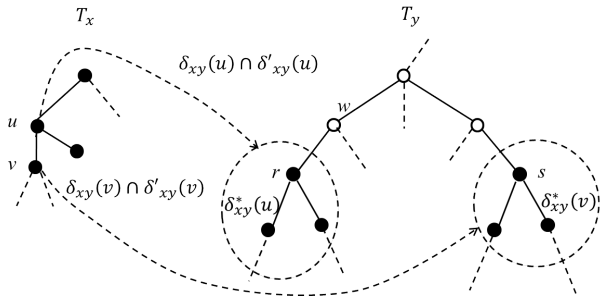


Fig. 4. Illustration of proof of Lemma 13.

At last, we give the main result of this section.

Theorem 4. *Let Δ be a tree-preserving constraint network. If no inconsistency is detected, then enforcing arc- and path-consistency determines the consistency of Δ and transforms Δ into a globally consistent network.*

Proof. If we can show that Δ is still tree-preserving after enforcing arc and path-consistency, then by Theorem 1 the new network is globally consistent if no inconsistency is detected.

By Proposition 3, Δ remains tree-preserving after enforcing arc-consistency. To enforce path-consistency on Δ, we need to call the following updating rule

$$\delta_{xy} \leftarrow \delta_{xy} \cap (\delta_{xz} \circ \delta_{zy}) \tag{2}$$

for $x, y, z \in V$ until the network is stable.

Suppose Δ is arc-consistent and tree-preserving w.r.t. trees T_x for $x \in V$ before applying (2). Note that if $\delta^* = \delta_{xy} \cap (\delta_{xz} \circ \delta_{zy})$ (as well as its converse) is arc-consistent, then $\delta^*(u)$ is nonempty for any node u in T_x. By Corollary 1, no branches or trunks need to be pruned in either T_x or T_y. Furthermore, by Lemma 13, $\delta^*(u) \cup \delta^*(v)$ is connected for every edge $\{u, v\}$ in T_x as there are no unsupported nodes in T_y under the converse of δ^*. Therefore δ^* is arc-consistent and consecutive, hence, tree-preserving.

If δ^* is not arc consistent, then we delete all unsupported values from T_x and T_y and enforce arc-consistency on Δ. If no inconsistency is detected then we have an updated arc-consistent and tree-preserving network by Lemma 11. Still write Δ for this network and recompute $\delta^* = \delta_{xy} \cap (\delta_{xz} \circ \delta_{zy})$ and repeat the above procedure until either inconsistency is detected or δ^* is arc-consistent. Note that, after enforcing arc-consistency, the composition $\delta_{xz} \circ \delta_{zy}$ may have changed.

Once arc-consistency of δ^* is achieved, we update δ_{xy} with δ^* and continue the process of enforcing path-consistency until Δ is path-consistent or an inconsistency is detected. □

5.2 Partial Path-Consistency

The partial path-consistency (PPC) algorithm was first proposed by Bliek and Sam-Haroud [2]. The idea is to enforce path consistency on sparse graphs by triangulating instead of completing them. Bliek and Sam-Haroud demonstrated that, as far as

CRC constraints are concerned, the pruning capacity of path consistency on triangulated graphs and their completion are identical on the common edges.

An undirected graph $G = (V, E)$ is *triangulated* or *chordal* if every cycle of length greater than 3 has a chord, i.e. an edge connecting two non-consecutive vertices of the cycle. For a constraint network $\Delta = \{v_i \delta_{ij} v_j : 1 \leq i, j \leq n\}$ over $V = \{v_1, ..., v_n\}$, the *constraint graph* of Δ is the undirected graph $G(\Delta) = (V, E(\Delta))$, for which we have $(v_i, v_j) \in E(\Delta)$ iff δ_{ij} is not a universal constraint. Given a constraint network Δ and a graph $G = (V, E)$, we say Δ is *partial path-consistent w.r.t.* G iff for any $1 \leq i, j, k \leq n$ with $(v_i, v_j), (v_j, v_k), (v_i, v_k) \in E$ we have $\delta_{ik} \subseteq \delta_{ij} \circ \delta_{jk}$ [2].

Theorem 5. *Let Δ be a tree-preserving constraint network. Suppose $G = (V, E)$ is a chordal graph such that $E(\Delta) \subseteq E$. Then enforcing partial path-consistency on G is equivalent to enforcing path-consistency on the completion of G, in the sense that the relations computed for the constraints in G are identical.*

Proof. The proof is similar to the one given for CRC constraints [2, Theorem 3]. This is because, (i) when enforcing arc- and path-consistency on a tree-preserving constraint network, in each step, we obtain a new tree-preserving constraint network; and (ii) path-consistent tree convex constraint networks are globally consistent. □

Remark 1. Note that our definition and results of tree-preserving constraints can be straightforwardly extended to domains with acyclic graph structures (which are connected or not). We call such a structure a *forest* domain. Given a tree-preserving constraint network Δ over forest domains $F_1, ..., F_n$ of variables $v_1, ..., v_n$. Suppose F_i consists of trees $t_{i,1}, ..., t_{i,k_i}$. Note that the image of each tree, say $t_{i,1}$, of F_i under constraint R_{ij} is a subtree t of F_j. Assume t is contained in the tree $t_{j,s}$ of forest F_j. Then the image of $t_{j,s}$ under constraint R_{ji} is a subtree of $t_{i,1}$. This establishes, for any $1 \leq i \neq j \leq n$, a 1-1 correspondence between trees in F_i and trees in F_j if the image of each tree is nonempty. In this way, the consistency of Δ is reduced to the consistency of several parallel tree-preserving networks over tree domains.

6 Tree-Preserving Constraints and the Scene Labelling Problem

The scene labelling problem [10] is a classification problem where all edges in a line-drawing picture have to be assigned a label describing them. The scene labelling problem is NP-complete in general. This is true even in the case of the trihedral scenes, i.e. scenes where no four planes share a point [12].

Labels used in the scene labelling problem are listed as follows:

'$+$' The edge is convex which has both of its corresponding planes visible;
'$-$' The edge is concave which has both of its corresponding planes visible;
'$\rightarrow$' Only one plane associated with the edge is visible, and when one moves in the direction indicated by the arrow, the pair of associated planes is *to the right*.

In the case of trihedral scenes, there are only four basic ways in which three plane surfaces can come together at a vertex [10]. A vertex projects in the picture into a 'V', 'W', 'Y' or 'T'-junction (each of these junction-types may appear with an arbitrary

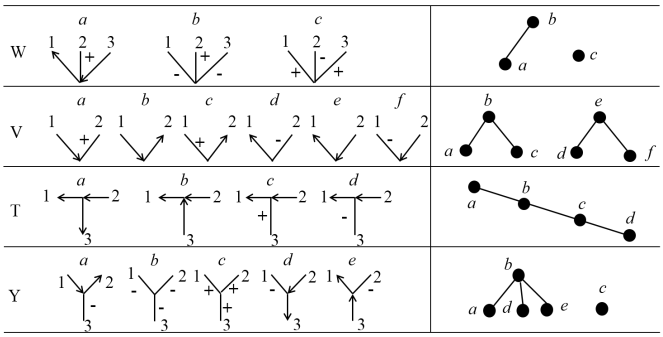


Fig. 5. Possible labelled line configurations of a junction in a picture and their corresponding forest structures.

rotation in a given picture). A complete list of the labelled line configurations that are possible in the vicinity of a node in a picture is given in Fig. 5.

In this section, we show that (i) every instance of the trihedral scene labelling problem can be modelled by a tree convex constraint network; (ii) a large subclass of the trihedral scene labelling problem can be modelled by tree-preserving constraints; (iii) there exists a scene labelling instance which can be modelled by tree-preserving constraints but not by chain- or CRC constraints.

A CSP for the scene labelling problem can be formulated as follows. Each junction in the line-drawing picture is a variable. The domains of the vertices are the possible configurations as shown in Fig. 5. The constraints between variables are simply that, if two variables share an edge, then the edge must be labeled the same at both ends.

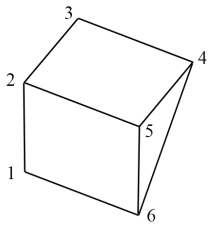
Proposition 5. *Every instance of the trihedral scene labelling problem can be modelled by a tree convex constraint network. Furthermore, there are only 39 possible configurations of two neighbouring nodes in 2D projected pictures of 3D trihedral scenes, and 29 out of these can be modelled by tree-preserving constraints.*

Proof. The complete list of these configurations and their corresponding tree convex or tree-preserving constraints is attached as an appendix. Note that we do not consider T-junctions in line drawing pictures since they decompose into unary constraints. □

As a consequence, we know that these 29 configurations of the scene labelling problem with 2D line-drawing pictures can be solved by the path-consistency algorithm in polynomial time. Moreover, since it is NP-hard to decide if a trihedral scene labelling instance is consistent, we have the following corollary.

Corollary 2. *The consistency problem of tree convex constraint networks is NP-complete.*

We next give a scene labelling instance which can be modelled by tree-preserving constraints but not by chain-preserving or CRC constraints. Consider the line drawing in the left of the following figure and the constraints for the drawing listed in the right. One can easily verify that all constraints are tree-preserving w.r.t. the forest structures listed in Fig. 5, but, for example, δ_{21} is not chain-preserving for the forest structures illustrated in Fig. 5 and δ_{25} is not CRC.



(*a*)

	a	*b*	*c*	*d*	*e*	*f*
a	0	0	0	1	1	0
b	0	0	0	0	0	1
c	0	0	1	0	0	0

δ_{21}

	a	*b*	*c*	*d*	*e*	*f*
a	0	0	0	0	1	1
b	0	0	0	1	0	0
c	1	0	0	0	0	0

δ_{23}

	a	*b*	*c*	*d*	*e*
a	0	0	1	0	0
b	0	0	1	0	0
c	0	1	0	1	0

δ_{25}

	a	*b*	*c*	*d*	*e*	*f*
a	0	0	0	1	1	0
b	0	0	0	0	0	1
c	0	0	1	0	0	0

δ_{43}

	a	*b*	*c*	*d*	*e*
a	0	0	1	0	0
b	0	0	1	0	0
c	0	1	0	0	1

δ_{45}

	a	*b*	*c*
a	1	0	0
b	0	1	0
c	0	0	1

δ_{46}

	a	*b*	*c*	*d*	*e*	*f*
a	0	0	0	0	1	1
b	0	0	0	1	0	0
c	1	0	0	0	0	0

δ_{61}

	a	*b*	*c*	*d*	*e*
a	0	0	1	0	0
b	0	0	1	0	0
c	1	1	0	0	0

δ_{65}

(*b*)

7 Further Discussion and Conclusion

In this paper, when formulating a CSP, we allow different variables have different tree domains. Feder and Vardi [7] and many other authors (see e.g. [11,1]) also considered CSPs which have a common domain D for all variables. These CSPs are called *one-sorted* in [3]. For one-sorted tree-preserving CSPs, we could also define a majority operator [11] under which the set of tree-preserving constraints is closed. This implies that the class of one-sorted tree-preserving CSPs has bounded strict width [7] and hence tractable. Indeed, such a majority operator ρ is defined as follows: for any three nodes a, b, c in a tree domain T, define $\rho(a, b, c)$ as the node d which is the intersection of paths $\pi_{a,b}$, $\pi_{a,c}$, and $\pi_{b,c}$. Following [3], it is straightforward to extend this result to multi-sorted tree-preserving CSPs.

In this paper, we identified two new tractable subclasses of tree convex constraint which are called path- and tree-preserving constraints, and proved that a chain- or path-preserving constraint network is in essence the disjoint union of several independent CRC constraint networks, and hence (re-)established the tractability of these constraints. More importantly, we proved that when enforcing arc- and path-consistency on a tree-preserving constraint network, in each step, the network remains tree-preserving. This implies that enforcing arc- and path-consistency will change a tree-preserving constraint network into a globally consistent constraint network. This also implies that the efficient partial path-consistent algorithm for large sparse networks is applicable for tree-preserving constraint network. As an application, we showed that a large class of the trihedral scene labelling problem can be modelled by tree-preserving constraints. This shows that tree-preserving constraints are useful in real world applications.

Acknowledgments

We sincerely thank the anonymous reviewers of CP-15 and IJCAI-15 for their very helpful comments. The majority operator was first pointed out to us by two reviewers

of IJCAI-15. This work was partially supported by ARC (FT0990811, DP120103758, DP120104159) and NSFC (61228305).

References

1. Barto, L., Kozik, M.: Constraint satisfaction problems solvable by local consistency methods. Journal of ACM 61(1), 3:1–3:19 (2014)
2. Bliek, C., Sam-Haroud, D.: Path consistency on triangulated constraint graphs. In: IJCAI-99. pp. 456–461 (1999)
3. Bulatov, A.A., Jeavons, P.: An algebraic approach to multi-sorted constraints. In: CP 2003. pp. 183–198 (2003)
4. Conitzer, V., Derryberry, J., Sandholm, T.: Combinatorial auctions with structured item graphs. In: AAAI'04. pp. 212–218 (2004)
5. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artificial Intelligence 49(1-3), 61–95 (1991)
6. Deville, Y., Barette, O., Hentenryck, P.V.: Constraint satisfaction over connected row convex constraints. Artificial Intelligence 109(1-2), 243–271 (1999)
7. Feder, T., Vardi, M.Y.: The computational structure of monotone monadic snp and constraint satisfaction: A study through datalog and group theory. SIAM Journal on Computing 28(1), 57–104 (1998)
8. Freuder, E.C.: Synthesizing constraint expressions. Communications of the ACM 21(11), 958–966 (1978)
9. Freuder, E.C.: A sufficient condition for backtrack-free search. Journal of the ACM 29(1), 24–32 (1982)
10. Huffman, D.A.: Impossible objects as nonsense sentences. Machine Intelligence 6(1), 295–323 (1971)
11. Jeavons, P., Cohen, D.A., Cooper, M.C.: Constraints, consistency and closure. Artificial Intelligence 101(1-2), 251–265 (1998)
12. Kirousis, L.M., Papadimitriou, C.H.: The complexity of recognizing polyhedral scenes. In: FOCS 1985. pp. 175–185 (1985)
13. Kumar, T.K.S.: Simple randomized algorithms for tractable row and tree convex constraints. In: AAAI'06. pp. 74–79 (2006)
14. Li, S., Liu, W., Wang, S.: Qualitative constraint satisfaction problems: An extended framework with landmarks. Artificial Intelligence 201, 32–58 (2013)
15. Maruyama, H.: Structural disambiguation with constraint propagation. In: ACL 1990. pp. 31–38 (1990)
16. Montanari, U.: Networks of constraints: Fundamental properties and applications to picture processing. Information Sciences 7, 95–132 (1974)
17. Van Beek, P., Dechter, R.: On the minimality and global consistency of row-convex constraint networks. Journal of the ACM 42(3), 543–561 (1995)
18. Zhang, Y., Freuder, E.C.: Properties of tree convex constraints. Artificial Intelligence 172(12-13), 1605–1612 (2008)
19. Zhang, Y., Marisetti, S.: Solving connected row convex constraints by variable elimination. Artificial Intelligence 173(12), 1204–1219 (2009)
20. Zhang, Y., Yap, R.H.C.: Consistency and set intersection. In: IJCAI-03. pp. 263–270 (2003)