Noname manuscript No.
(will be inserted by the editor)

On Tree-Preserving Constraints

Shufeng Kong · Sanjiang Li · Yongming Li · Zhiguo Long

Received: date / Accepted: date

Abstract The study of tractable subclasses of constraint satisfaction problems is a central topic in constraint solving. 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. However, it is NP-complete to decide whether a tree convex constraint network has solutions. This paper studies and compares three subclasses of tree convex constraints, which are called chain-, path-, and tree-preserving constraints respectively. The class of tree-preserving constraints strictly contains the subclasses of path-preserving and arc-consistent chain-preserving constraints. We prove that, when enforcing strong path-consistency on a tree-preserving constraint network, in each step, the network remains tree-preserving. This ensures the global consistency of consistent tree-preserving networks after enforcing strong path-consistency, and also guarantees the applicability of the partial path-consistency algorithms to tree-preserving constraint networks, which is usually much more efficient than the path-consistency algorithms for large sparse constraint networks. As an application, we show that the class of tree-preserving constraints is useful in solving the scene labelling problem.

Keywords Tree-preserving constraint · connected row convex constraint · scene labelling problem

1 Introduction

Constraint satisfaction problems (CSPs) have been widely used in many areas, such as scene labelling [19], natural language parsing [33], picture processing [34], and spatial and temporal reasoning [14,31]. Since deciding the consistency of CSP instances is NP-complete in general, lots of efforts have been devoted to identifying tractable subclasses. There are

Shufeng Kong, Sanjiang Li and Zhiguo Long
QSI, FEIT, University of Technology Sydney, Australia
E-mail: {Shufeng.Kong, Zhiguo.Long}@student.uts.edu.au, Sanjiang.Li@uts.edu.au

Yongming Li
College of Computer Science, Shaanxi Normal University, China
E-mail: liyongm@snnu.edu.cn

two main approaches for constructing tractable subclasses. The first approach is *structural-based*, in which tractable subclasses are obtained by restricting the topology of the underlying graph of the constraint network (being a tree or having treewidth bounded by a constant [13]); the second approach is *language-based*, in which tractable subclasses are obtained by by restricting the type of the allowed constraints between variables (cf. [35]). Recently, researchers also propose a hybrid approach for constructing tractable classes, see e.g., the subclass of CSP instances satisfying the *broken-triangle property* (BTP) [10, 11].

In this paper, we are mainly interested in the language-based tractable subclasses. Montanari [34] showed that path-consistency is sufficient to guarantee that a constraint network is globally consistent if the relations are all *monotone*. Van Beek and Dechter [35] generalised monotone constraints to *row convex* constraints, which are further generalised to *tree convex* constraints by Zhang and Yap [38]. These constraints also have the nice property that every path-consistent constraint network is globally consistent.

However, neither row convex constraints nor tree convex constraints are closed under composition and intersection, which are the main operations of *path-consistency* (PC) algorithms. This means that enforcing path-consistency may destroy row and tree convexity. Deville et al. [15] proposed a tractable subclass of row convex constraints, called *connected row convex* (CRC) constraints, which are closed under composition and intersection. Zhang and Freuder [37] also identified a tractable subclass of tree convex constraints, called *locally chain convex and strictly union closed* constraints. They also proposed the important notion of *consecutive* constraints. Kumar [27] showed that the subclass of arc-consistent consecutive tree convex (ACCTC) constraints is tractable by providing a polynomial time randomised algorithm. Nevertheless, for the ACCTC problems, "*it is not known whether there are efficient deterministic algorithms, neither is it known whether strong path-consistency ensures global consistency on those problems*."[37]

In this paper, we study and compare three subclasses of tree convex constraints which are called, respectively, chain-, path- and tree-preserving constraints.

In Section 2, we start with basic notations and concepts that will be used throughout the paper. Based on the concept of tree domains, we introduce chain-, path- and tree-preserving constraints. Chain-preserving constraints are exactly "locally chain convex and strictly union closed" constraints in the sense of [37], which include CRC constraints as a special case where tree domains are linear. Arc-consistent chain-preserving constraints, path-preserving constraints and ACCTC constraints are all strictly contained in the class of tree-preserving constraints.

Therefore, the remainder of this paper will focus on the more general tree-preserving constraints. We show in Section 3 that the class of tree-preserving constrains is closed under intersection and composition, which are operations of the path-consistency algorithm. This guarantees that a tree-preserving constraint network remains tree-preserving after enforcing path-consistency on it. Recall that every path-consistent tree convex constraint network is globally consistent [38]. This shows that the class of tree-preserving constraints is tractable and can be solved by the path-consistency algorithm. We also prove in this section that our definitions and results for tree-preserving constraints can be extended to domains with acyclic graph structures, called *forest domains* in this paper.

The above properties of tree-preserving constraints bear similarity to CRC constraints. Bliek and Sam-Haroud [4] showed that enforcing *partial path-consistency* (PPC) is sufficient to solve sparse CRC constraint networks. PPC enforces PC on sparse constraint graphs by triangulating instead of completing them and thus can be enforced more efficiently than enforcing PC. 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. In Section 4, we show that PPC [4] is sufficient to decide the consistency of tree-preserving constraint networks. Moreover, we show that, after enforcing PPC, we can find a solution in a backtrack-free style if no inconsistency is detected.

Section 5 is concerned with the application of tree-preserving constraints in scene labelling. Solving the scene labelling problem is a crucial part of figuring out the possible 3D scenes of a 2D projection, which has applications in both vision and geometric modelling. Research in this field has centred on the trihedral scene labelling problem, i.e. scenes where no four planes share a point. The trihedral scene labelling problem has been shown to be NP-complete [23]. Based on the forest domains associated to each possible variable type by Zhang and Freuder [37] (see Fig. 9), we show that all 39 possible types of the trihedral scene labelling problem instances are tree convex, and 29 of them are tree-preserving. This means that a large subclass of the trihedral scene labelling problem can be modelled by tree-preserving constraint networks and thus can be efficiently solved by the techniques discussed in this paper. As a byproduct, since every instance of the NP-complete trihedral scene labelling problem can be modelled by a tree convex constraint network, we show that the class of tree convex constraints is NP-complete.

It is interesting to compare our approach with another research line of studying tractable subclasses of CSPs, which focuses on the algebraic closure property of constraints [5, 16, 20]. In Section 6, we study the algebraic closure property of tree-preserving constraints and establish the equivalence between tree-preserving constraints and constraints that are closed under a "standard" majority operation. In this way, we provide an alternative way to prove the tractability of tree-preserving constraints.

Section 7 reports experimental evaluations on enforcing PPC and PC on tree-preserving constraint networks and Section 8 concludes the paper.

This paper is a significant extension of the conference paper [26]. Apart from detailed proofs for most of our results and more figures for illustrating the key notions and techniques, we make the following major extensions: (i) We add Section 4 to show that enforcing PPC will enable us to find a solution for a tree-preserving constraint network in a backtrack-free style. By this result, we also present an efficient algorithm based upon PPC for finding a solution of a tree-preserving constraint network. (ii) We add Section 6 to discuss the algebraic closure property of tree-preserving constraints. Actually, we prove that, given a relation δ between two tree domains T_x and T_y, if both δ and its inverse are arc-consistent, then δ is closed under the "standard" majority operations (on T_x and T_y) *if and only if* δ and its inverse are both tree-preserving (w.r.t. T_x and T_y). (iii) We add Section 7 to report experimental evaluations of local consistency enforcing algorithms for sparse tree-preserving constraint networks.

2 Preliminaries

Let D be the domain of a variable x. An undirected 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 labelling [37] and combinatorial auctions [8]. We note that, in this paper, we fix a specific tree domain for each variable x.

In this paper, we distinguish between trees and rooted trees. 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 neighbours of a in G. A node a is called a *leaf node* if it has only one neighbour, i.e. $\deg(a) = 1$.

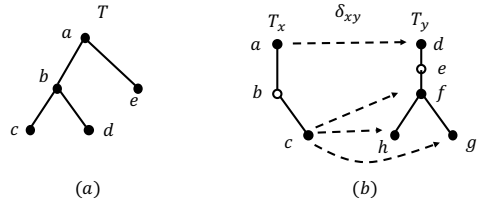

Fig. 1 (a) A tree T. Subgraphs with node sets $\{a, b, c, d\}$, $\{a, b, c\}$, and $\{c, b, d\}$ form, respectively, a subtree, a chain, and a path of T. The degree of node b is 3 and the three branches of b are the subtrees with node sets $\{c\}$, $\{d\}$ and $\{a, e\}$ respectively. (b) A binary constraint δ_{xy} between T_x and T_y, where a dashed arrow from a node u in T_x to a node v in T_y indicates that (u, v) is in δ_{xy}. Node c is supported under δ_{xy} with image $\{f, g, h\}$, and b is unsupported under δ_{xy}.

Definition 1 A *tree* is a connected graph without any cycle (cf. Fig. 1(a)). 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. The empty subgraph is a subtree of any tree.

Let T be a tree (rooted tree, resp.) and I a subtree of T. I is a *path* (*chain*, resp.) in T if each node in I has at most two neighbours (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 $T \setminus \{a\}$.

Fig. 1 gives illustrations of these notions.

Throughout this paper, we always associate a tree structure $T_x = (D_x, E_x)$ with a given domain D_x, where E_x is the set of tree edges connecting values in D_x. For convenience, we often use the notation T_x to denote the domain D_x and call T_x a tree domain. Also, $a \in T_x$ means that $a \in D_x$.

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, i.e. $\delta \subseteq D_x \times D_y$. For simplicity, we often denote this constraint by δ. A value $u \in D_x$ is *supported* under δ if there exists a value v in D_y s.t. $\langle u, v\rangle \in \delta$. 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 $\delta(A) = \{b \in D_y : (\exists a \in A)\langle a, b\rangle \in \delta\}$. For $A = \{a\}$, without confusion, we also use $\delta(a)$ to represent $\delta(\{a\})$ (cf. Fig. 1(b)).

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, unless stated otherwise, *we assume that δ_{xy} is the inverse of δ_{yx}*, and if there is no constraint for (x, y), we assume that δ_{xy} is the universal constraint (i.e. $D_x \times D_y$).

Definition 3 [17, 18] A constraint network Δ over n variables is *k-consistent* if 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* if 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-consistency* (AC) and *path-consistency* (PC) respectively.

Specially, we call a constraint network *strongly path-consistent* if it is both arc-consistent and path-consistent, and call a binary constraint δ_{xy} *arc-consistent* if for any $a \in D_x$, there is some $b \in D_y$ such that $\langle a, b\rangle \in \delta_{xy}$.

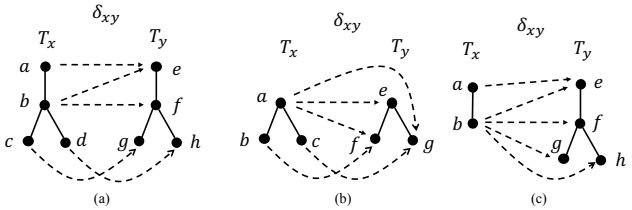

Fig. 2 (a) δ_{xy} is a chain- but not path-preserving constraint as the the image of the path $\{c, b, d\}$ of T_x is $\{e, f, g, h\}$, which is not a path of T_y; (b) δ_{xy} is a path- but not chain-preserving constraint as the image of the chain $\{a, c\}$ of T_x is $\{e, f, g\}$, which is not a chain of T_y; (c) δ_{xy} is a tree-preserving but neither path- nor chain-preserving constraint as the image of the path $\{a, b\}$ of T_x, which is also a chain of T_x, is $\{e, f, g, h\}$ that is neither a path nor a chain of T_y.

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 (cf. Fig. 2)

- *tree convex* if the image of every value a in D_x (i.e. $\delta(a)$) is a 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.

We note that a subtree (a path or a chain) of T_x (or T_y) in the above definition is possibly empty. We also note that chain-preserving constraints are exactly those "locally chain convex and strictly union closed" constraints defined in [37].

Connected row convex (CRC) constraints are special chain-preserving constraints defined over chain domains. The following definition of CRC constraints is equivalent to the one given in [15].

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 [15].

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.

The following proposition summarises relations between these tree convex constraints.

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.*

Proof First, we notice that an edge is a chain, a path and a subtree, and a chain or a path is a subtree. Then the claim that every chain-, path-, or tree-preserving constraint is consecutive directly follows from Definition 4.

Second, we show that if a constraint δ_{xy} is path-preserving, then it is also tree-preserving. Let δ_{xy} be a path-preserving constraint over tree domains T_x and T_y. Suppose that δ_{xy} is not tree-preserving. We know that there exists a subtree t_x of T_x such that $\delta_{xy}(t_x)$ is not a subtree of T_y. Therefore, t_x can be divided into two parts, say t_x^1 and t_x^2, such that $\delta_{xy}(t_x^1)$ is disconnected from $\delta_{xy}(t_x^2)$. Let $v_1 \in t_x^1$, $v_2 \in t_x^2$ such that $\delta_{xy}(v_1) \neq \varnothing$ and $\delta_{xy}(v_2) \neq \varnothing$. Let π_{v_1,v_2} be the path from v_1 to v_2 in T_x. Let $\pi_1 = \pi_{v_1,v_2} \cap t_x^1$ and $\pi_2 = \pi_{v_1,v_2} \cap t_x^2$. Then $\delta_{xy}(\pi_1)$ is disconnected from $\delta_{xy}(\pi_2)$. Therefore, $\delta_{xy}(\pi_{v_1,v_2})$ cannot be a path of T_y which contradicts that δ_{xy} is path-preserving.

Finally, if a constraint δ_{xy} over tree domains T_x and T_y is arc-consistent consecutive tree convex, we show that δ_{xy} is also tree-preserving. Let t_x be an arbitrary subtree of T_x. We show that $\delta_{xy}(t_x)$ is a subtree of T_y. Let t_0 be a subtree of t_x such that $|t_0| = 1$. Because δ_{xy} is tree convex, $\delta_{xy}(t_0)$ is a subtree of T_y. Let t be a subtree of t_x and $e_{v_1v_2}$ be an edge of T_x with $v_1 \in t$, $v_2 \notin t$ and $v_2 \in t_x$. Suppose $\delta_{xy}(t)$ is a subtree of T_y. Because δ_{xy} is arc-consistent consecutive tree-convex, $\delta_{xy}(e_{v_1v_2})$ is also a subtree of T_y, and $\delta_{xy}(v_1) \neq \varnothing$. Because $\varnothing \neq \delta_{xy}(v_1) \subseteq \delta_{xy}(t) \cap \delta_{xy}(e_{v_1v_2})$, $\delta_{xy}(t)$ and $\delta_{xy}(e_{v_1v_2})$ are connected. Thus, $\delta_{xy}(t \cup e_{v_1v_2})$ is also subtree of T_y. Therefore, by induction, we can add edges to t_0 one by one until $t_0 = t_x$, and in each step, $\delta_{xy}(t_0)$ is a subtree of T_y. □

Although every arc-consistent chain- or path-preserving constraint is tree-preserving, Fig. 2(c) shows that the other direction is not always true. Furthermore, Fig. 1(b) shows that not every chain-preserving (or consecutive tree convex) constraint is tree-preserving and Fig. 2(a,b) show that chain-preserving constraints and path-preserving constraints are incomparable.

The following results of trees will be used in the proof of some results in our paper.

Lemma 1 *[38] Let T be a tree and suppose t_i $(i = 1, \ldots, m)$ are subtrees of T. Then $\bigcap_{i=1}^{m} t_i$ is nonempty iff $t_i \cap t_j$ is nonempty for every $1 \leq i \neq j \leq m$.*

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 $t \cap t' = \varnothing$.*

Proof The first part is clear as the edge $\{u, v\}$ connects t and t'. Suppose $u \notin t'$, $v \notin t$. We show $t \cap t' = \varnothing$. Suppose this is not the case and there exists $w \in t \cap t'$. Then we have $\pi_{w,u} \subseteq t$ and $\pi_{w,v} \subseteq t'$. Since u is a neighbour of v, we have either $u \in \pi_{w,v}$ or $v \in \pi_{w,u}$, i.e. either $u \in t'$ or $v \in t$. Both contradict our assumption that $u \notin t'$, $v \notin t$. Therefore, we must have $t \cap t' = \varnothing$. □

Using Lemma 1, Zhang and Yap [38] proved the following result:

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

In the following, we will focus on the class of tree-preserving constraints.

3 Tree-Preserving Constraints

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 strong path-consistency on Δ, in each step, the network remains tree-preserving. Hence, by Theorem 1, we know that, if no inconsistency is detected, a tree-preserving constraint network will be transformed into an equivalent globally consistent network after enforcing strong path-consistency.

Firstly, we show that tree-preserving constraint networks are closed under arc-consistency.

Lemma 3 *Suppose δ_{xy} and δ_{yx} are tree-preserving (tree convex) w.r.t. tree domains T_x and T_y. Let t be a subtree of T_x and $\delta'_{xy} = \{\langle a, b\rangle \in \delta_{xy} : a \in t\}$ and $\delta'_{yx} = \{\langle b, a\rangle \in \delta_{yx} : a \in t\}$ the restrictions of δ_{xy} and δ_{yx} to t. Then both δ'_{xy} and δ'_{yx} are tree-preserving (tree convex).*

Proof Note that a path or subtree of t is also a path or subtree of T_x. The conclusion then follows directly from the definitions of tree-preserving and tree convex constraints. □

As a corollary, we have

Corollary 1 *Let Δ be a tree-preserving (tree convex) constraint network over tree domains T_x ($x \in V$). Assume that t is a nonempty subtree of T_x. When restricted to t, Δ remains tree-preserving (tree convex).*

The following lemma examines unsupported values of a tree-preserving constraint.

Lemma 4 *Suppose δ_{xy} is nonempty and 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 tree-preserving, $\delta_{xy}(\pi_{u_1,u_2})$ is a subtree 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 tree-preserving.

The following result then follows directly.

Proposition 2 *Let Δ be a tree-preserving constraint network over tree domains T_x ($x \in V$). If no inconsistency is detected, then Δ remains 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 4, 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 Corollary 1, the restricted constraint network to t remains tree-preserving. □

Secondly, we consider the intersection and composition of tree-preserving constraints.

When doing relational intersection, we may need to remove some unsupported values from domains. Unlike CRC [15, Lemma 13] and chain-preserving constraints [37, Proposition 5], 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 7 Suppose $a \neq b$ are two nodes of a tree T that are not neighbours. The *trunk* between a, b, written as $M_{a,b}$, is defined as the connected component of $T \setminus \{a, b\}$ which contains all the internal nodes of $\pi_{a,b}$ (see Fig. 3). The *M-contraction* of T by $M_{a,b}$, denoted by $T \ominus M_{a,b}$, is the tree obtained by removing the nodes with associated edges in $M_{a,b}$ and adding an edge $\{a, b\}$ to T.

To improve readability, we defer the proofs of Lemmas 5-7 to Appendix.

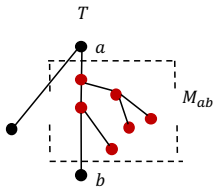

Fig. 3 M_{ab} is a trunk of tree T, i.e., the subtree induced by vertices with red colour.

Lemma 5 *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 no inconsistency is detected.*

The following two lemmas consider the intersection of two tree-preserving constraints.

Lemma 6 *Assume δ_{xy} and δ'_{xy} are two arc-consistent and tree-preserving constraints w.r.t. trees T_x and T_y. Let $\delta^*_{xy} = \delta_{xy} \cap \delta'_{xy}$. Let $W = \{w \in T_x \mid \delta^*_{xy}(w) \neq \varnothing\}$ be the set of supported values of δ^*_{xy}. Suppose $u \in T_x$ and $u \notin W$. Then there exist at most two values w_1, w_2 in W s.t. no value in W other than w_i is on the path $\pi_{w_i,u}$ for $i = 1, 2$.*

Remark 1 If there is only one value w_1 in W that satisfies the condition in Lemma 6, regarding u as the root of T_x, then W is contained in the subtree rooted at w_1; if there are two values w_1, w_2 in W that satisfy the condition in the Lemma 6, then the trunk M_{w_1,w_2} should be contracted from T_x to make W connected.

Lemma 7 *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. $\delta^*_{xy}(u) \neq \varnothing$, $\delta^*_{xy}(v) \neq \varnothing$, and $\delta^*_{xy}(u) \cup \delta^*_{xy}(v)$ is disconnected in T_y. Then there exist unique $r \in \delta^*_{xy}(u)$ and $s \in \delta^*_{xy}(v)$ s.t. every node in $M_{r,s}$ is unsupported under δ^*_{yx}.*

The following result follows from the definition of tree-preserving constraints.

Proposition 3 *Assume that 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.*

Proof Let $\delta'_{xy} = \delta_{xz} \circ \delta_{zy}$ and t_x be an arbitrary subtree of T_x. Then we have that $\delta'_{xy}(t_x) = \delta_{zy}(\delta_{xz}(t_x))$. Because δ_{xz} is tree-preserving, we have that $\delta_{xz}(t_x)$ is a subtree of T_z. Similarly, $\delta_{zy}(\delta_{xz}(t_x))$ is a subtree of T_y. Thus, δ'_{xy} is tree-preserving. □

Finally, we give the main result of this section.

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

Proof Fig. 4 is the flow diagram of the proof.

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

By Proposition 2, Δ remains tree-preserving after enforcing arc-consistency. To enforce path-consistency on Δ, we need to call the following updating rule:

$$\delta_{xy} \leftarrow \delta_{xy} \cap (\delta_{xz} \circ \delta_{zy}) \tag{1}$$

$$\delta_{yx} \leftarrow \delta_{xy}^{-1} \tag{2}$$

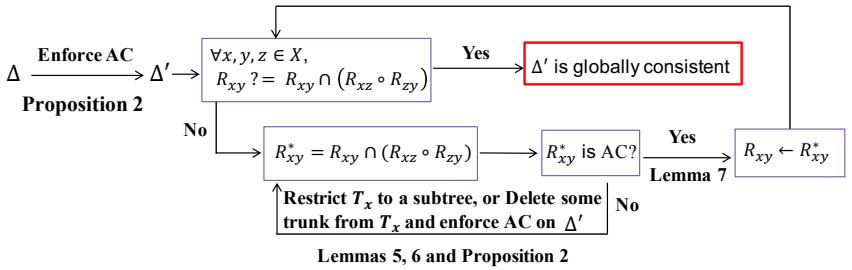

Fig. 4 The flow diagram of proof of Theorem 2.

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 (1). Note that if $\delta^*_{xy} = \delta_{xy} \cap (\delta_{xz} \circ \delta_{zy})$ (as well as its inverse δ^*_{yx}) is arc-consistent, then $\delta^*_{xy}(u)$ is nonempty for any node u in T_x. By Lemma 7, $\delta^*_{xy}(u) \cup \delta^*_{xy}(v)$ is connected for every edge $\{u, v\}$ in T_x as otherwise there will exist unsupported nodes in T_y under the inverse of δ^*_{xy}. Therefore δ^*_{xy} is arc-consistent and consecutive, and hence, tree-preserving. Since $\delta^*_{yx} = \delta^{*-1}_{xy} = \delta_{yx} \cap (\delta_{yz} \circ \delta_{zx})$, analogously, we have δ^*_{yx} is tree-preserving.

If δ^*_{xy} is not arc-consistent, then there exists $u \in T_x$ s.t. $\delta^*_{xy}(u)$ is empty. By Lemma 6 and Remark 1, we should restrict the domain to a subtree or contract some trunk from T_x and enforce arc-consistency. If δ^*_{yx} is not arc-consistent, then we do analogously. By Lemma 5 and Proposition 2, if no inconsistency is detected, then we have an updated arc-consistent and tree-preserving network. Still write Δ for this network and recompute $\delta^*_{xy}, \delta^*_{yx}$ and repeat the above procedure until either an inconsistency is detected or both δ^*_{xy} and δ^*_{yx} are arc-consistent. Note that, after enforcing arc-consistency, the composition $\delta_{xz} \circ \delta_{zy}$ may have changed.

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

In above, we assume that each domain is associated to a tree structure. Actually, our definitions 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.

Proposition 4 *The consistency of a tree-preserving constraint network over forest domains can be reduced to the consistency of several parallel tree-preserving networks over tree domains.*

Proof Given a tree-preserving constraint network Δ over forest domains $F_1, ..., F_n$ of variables $v_1, ..., v_n$, suppose that F_i consists of trees (i.e., maximally connected components) $t_{i,1}, \ldots, t_{i,k_i}$. Note that the image of each tree, say $t_{i,1}$, of F_i under constraint δ_{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 δ_{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. □

Fig. 5 shows a tree-preserving network over forest domains. We note that Δ cannot be modelled as tree-preserving over tree domains. For example, if we modify F_1 as a tree T_1

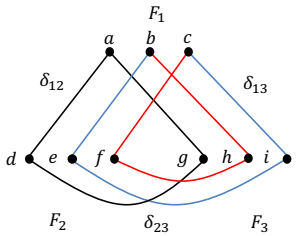

Fig. 5 A tree-preserving network $\Delta = \{\delta_{12}, \delta_{13}, \delta_{23}\}$ over forest domains F_1, F_2, and F_3, where each forest domain consists of three trees which contain only one node.

by adding edges $\{a, b\}$ and $\{b, c\}$. Then, in order to make δ_{13} tree-preserving, edges $\{g, h\}$ and $\{h, i\}$ should be added to F_3. Write T_3 for the new tree. Likewise, in order to make δ_{12} tree-preserving, edges $\{d, e\}$ and $\{e, f\}$ should be added to F_2. Write T_2 for the new tree. However, δ_{23} is not tree-preserving w.r.t. T_2 and T_3.

Recall that when enforcing path-consistency, we transform a constraint network into a complete constraint graph despite the number of non-trivial constraints it has. In the following section, we consider a more efficient path-consistency algorithm that respects the density of non-trivial constraints in the network.

4 Partial Path-Consistency

Partial path-consistency (PPC) [4] is a more general consistency condition than PC and can be enforced more efficiently for constraint networks with sparse constraint graphs. The idea of PPC is to enforce path-consistency on sparse constraint 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 completions are identical on the common edges. In this section, we show that a similar result applies to tree-preserving constraints. Moreover, we show that, after enforcing strong PPC (i.e. both AC and PPC), we can find a solution in a *backtrack-free* style if no inconsistency is detected.

We first recall some basic definitions and results related to graph triangulation. 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.

Definition 8 (cf. e.g. [4]) Given a graph $G = (V, E)$, the *neighbourhood* of vertex v in V is $N(v) = \{w \in V \mid (v, w) \in E\}$. A vertex v of G is a *simplicial vertex* if the induced subgraph of $N(v)$ is complete. A *perfect vertex elimination ordering* of G is an ordering $\langle v_1, v_2, ..., v_n \rangle$ such that for $1 \leq i \leq n-1$, v_i is a simplicial vertex of the subgraph of G induced by $\{v_i, v_{i+1}, ..., v_n\}$.

Proposition 5 ([24]) *Every triangulated graph has a simplicial vertex, and a graph is triangulated iff it has a perfect vertex elimination ordering.*

Definition 9 ([4]) Suppose $\prec = \langle v_1, v_2, ..., v_n \rangle$ is a perfect vertex elimination ordering of graph G. For $1 \leq i \leq n$, we denote by G_i the subgraph of G induced by $S_i = \{v_{n-i+1}, ..., v_n\}$ and write $F_i = \{v_k \in N(v_{n-i}) | v_{n-i} \prec v_k\}$.

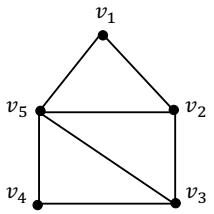

Fig. 6 A triangulated graph G, where $\langle v_1, v_2, v_3, v_4, v_5\rangle$ is a perfect vertex elimination ordering of G.

Since the vertex elimination ordering is perfect, the subgraph induced by F_i is complete. An example is given in Fig. 6, where G is a triangulated constraint graph and $\langle v_1, \ldots, v_5\rangle$ is a perfect vertex elimination ordering of G. By Definition 9, we can see that S_i just denotes the last i vertices w.r.t. the ordering. Therefore, we have $S_1 = \{v_5\}, S_2 = \{v_4, v_5\}, S_3 = \{v_4, v_4, v_5\}, S_4 = \{v_2, v_3, v_4, v_5\}$ and $S_5 = \{v_1, v_2, v_3, v_4, v_5\}$. Finally, F_i denotes the set of vertices that are adjacent to v_{n-i} and after it w.r.t. the ordering. Therefore, we have $F_1 = \{v_5\}, F_2 = \{v_4, v_5\}, F_3 = \{v_3, v_5\}$ and $F_4 = \{v_2, v_5\}$.

For a constraint network Δ 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 $\delta_{v_i v_j} \in \Delta$. Note that here we do not assume that there is a constraint between every pair of variables.

Let $\pi = \langle x = u_0, \cdots, u_i, \cdots, u_k = y\rangle$ be a path in G with $\{x, y\} \in E(\Delta)$. We say π is *path-consistent* (PC) if for all (c_0, c_k) in δ_{xy}, we can find values for the intermediate variables u_i $(0 < i < k)$ such that all the constraints $\delta_{u_i,u_{i+1}}$ $(0 \leq i < k)$ are satisfied. Note that if δ_{xy} is empty for some x, y, then π is regarded as not path-consistent in this paper. A constraint graph G is PC iff all paths in G are PC [32]. Note that this constraint graph-based definition of PC is equivalent to the one given in Definition 3 when G is a complete graph.

In this section, we make a distinction between enforcing PC on a constraint network Δ and on its constraint graph $G(\Delta)$. It is clear that a constraint network is PC if the completion of its constraint graph is PC. We say a constraint network Δ is *partially path-consistent* (PPC) if its constraint graph $G(\Delta)$ is PC. The following result shows that we only need to consider paths of length 2 if the constraint graph is triangulated.

Proposition 6 ([4]) *A triangulated constraint graph G is path-consistent iff every path of length 2 is path-consistent.*

The following result is a straightforward extension from CRC constraints [4] to tree-preserving constraints. For the purpose of being self-contained, we provide a complete proof below.

Lemma 8 *Suppose Δ is a strongly partial path-consistent tree-preserving constraint network with a triangulated constraint graph $G = (V, E)$. Assume that $\prec = \langle v_1, v_2, ..., v_n\rangle$ is a perfect vertex elimination ordering of G. Let $G_i, S_i, F_i (1 \leq i \leq n)$ be defined as in Definition 9. Suppose i is the largest index such that G_i is complete. Assume that v_j is a node in S_i that is not a neighbour of v_{n-i} in G. Let $\delta_{v_{n-i},v_j} = \bigcap_{x \in F_i} \left(\delta_{v_{n-i},x} \circ \delta_{x,v_j}\right)$.*

Then

(i) The constraint graph $G' = (V, E \cup \{(v_{n-i}, v_j)\})$ is triangulated and $\prec$ is also a perfect vertex elimination ordering of G' and F_i is exactly the set $\{x \mid (v_{n-i}, x), (x, v_j) \in E\}$;
(ii) G' is strongly path-consistent;

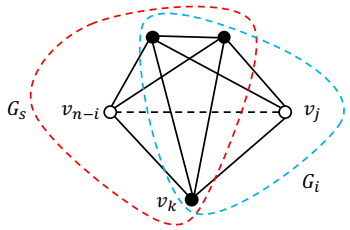

Fig. 7 Illustration of proof of Lemma 8. The figure is taken from [4].

(iii) The constraint δ_{v_{n-i},v_j} and its inverse are tree-preserving.

Proof (i) This result directly follows from the proof of [4, Lemma 1].

(ii) First, we show that G' is PC. To this end, because G' is triangulated, it is sufficient to show that every path of length 2 of G' is PC by Proposition 6. Because G is PC, paths of length 2 of G' that do not go through v_{n-i} and v_j are PC. So, let us consider paths of length 2 of G' that go through v_{n-i} and v_j.

Let G_s be the subgraph of G that is induced by $F_i \cup \{v_{n-i}\}$. Note that G_i is the subgraph of G that is induced by $F_i \cup \{v_j\}$. See Fig. 7 for an illustration where the vertices in F_i are colored black. We claim that G_s and G_i are globally consistent. By assumption, we know that G_i is complete. Because $\prec$ is a perfect vertex elimination ordering of G, we know that G_s is also complete. Now, because G_i and G_s are complete, strongly path-consistent and tree-preserving, they are globally consistent by Theorem 1.

Let π be a path of length 2 of G' that goes through v_{n-i} and v_j. We now show that π is PC. To this end, we have to consider the following two cases:

Case 1: $\pi = \langle v_{n-i}, v_k, v_j \rangle$ with some $v_k \in V (v_k \neq v_{n-i}, v_j)$. By (i), we know that F_i is exactly the set $\{x \mid (v_{n-i}, x), (x, v_j) \in E\}$. Therefore, $v_k \in F_i$. Because G_s is globally consistent, it admits at lease one solution, say α. Because G_i is also globally consistent, $\alpha|_{F_i}$ can be consistently extended to v_j such that all constraints in G_i are satisfied. Therefore, α can be extended to a solution of $G_s \cup G_i$, say β. Then we have that, for all $x \in F_i$, $\langle \beta|_{v_{n-i}}, \beta|_x \rangle \in \delta_{v_{n-i},x}$ and $\langle \beta|_x, \beta|_{v_j} \rangle \in \delta_{x,v_j}$. Therefore, $\langle \beta|_{v_{n-i}}, \beta|_{v_j} \rangle \in \bigcap_{x \in F_i} \left(\delta_{v_{n-i},x} \circ \delta_{x,v_j}\right)$. Thus, $\delta_{v_{n-i},v_j} = \bigcap_{x \in F_i} \left(\delta_{v_{n-i},x} \circ \delta_{x,v_j}\right) \neq \varnothing$. Further, because $v_k \in F_i$, for any $\langle a, b \rangle \in \delta_{v_{n-i},v_j}$, there is some $c \in D_k$ such that $\langle a, c \rangle \in \delta_{v_{n-i},v_k}$ and $\langle c, b \rangle \in \delta_{v_k,v_j}$, and thus $\pi = \langle v_{n-i}, v_k, v_j \rangle$ is PC.

Case 2: $\pi = \langle v_{n-i}, v_j, v_k \rangle$ with some $v_k \in V (v_k \neq v_{n-i}, v_j)$. Since G' is triangulated by (i), there is an edge $\{v_{n-i}, v_k\}$ of G'. Therefore, $v_k \in F_i$. Surely, δ_{v_{n-i},v_k} is not empty, because it is a constraint in G and G is PC. Now, we show that for any $\langle a, b \rangle \in \delta_{v_{n-i},v_k}$, there is some $c \in D_j$ such that $\langle a, c \rangle \in \delta_{v_{n-i},v_j}$ and $\langle c, b \rangle \in \delta_{v_j v_k}$. Because G_s is globally consistent, $\langle a, b \rangle$ can be extended to a solution of G_s. Similar to case 1, we know that the solution of G_s can be extended to a solution of $G_s \cup G_i$. We write such a solution of $G_s \cup G_i$ as ψ. Let value c be the restriction of ψ to variable v_j. Certainly, $\langle c, b \rangle \in \delta_{v_j v_k}$. Further, for any $x \in F_i$, let c' be the restriction of ψ to x. We have that $\langle a, c' \rangle \in \delta_{v_{n-i},x}$ and $\langle c', c \rangle \in \delta_{x,v_j}$, and thus $\langle a, c \rangle \in \delta_{v_{n-i},v_j} = \bigcap_{x \in F_i} \left(\delta_{v_{n-i},x} \circ \delta_{x,v_j}\right)$. Therefore, we have that $\pi = \langle v_{n-i}, v_j, v_k \rangle$ is also PC.

Second, we show that G' is AC. To this end, it is sufficient to prove that δ_{v_{n-i},v_j} is AC. Because $\delta_{v_{n-i},v_k} (v_k \in F_i)$ is AC, for any $a \in D_{n-i}$, there is a $v \in D_k$ such that $\langle a, b \rangle \in \delta_{v_{n-i},v_k}$. Further, because $\pi = \langle v_{n-i}, v_j, v_k \rangle$ is PC, there is a $c \in D_j$ such that

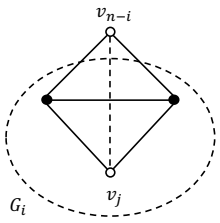

Fig. 8 Illustration of proof of Theorem 3.

$\langle a, c\rangle \in \delta_{v_{n-i},v_j}$ and $\langle c, b\rangle \in \delta_{v_j,v_k}$. Similarly, for any $c \in D_j$, there is a $a \in D_{n-i}$ such that $\langle c, a\rangle \in \delta_{v_j,v_{n-i}}$. So, δ_{v_{n-i},v_j} is AC.

(iii) By (i), we have $F_i = \{x \mid \{v_{n-i}, x\}, \{x, v_j\} \in E\}$. According to Proposition 3, we know that $\delta_{v_{n-i},x} \circ \delta_{x,v_j}$ is tree-preserving for any $x \in F_i$. Furthermore, because δ_{v_{n-i},v_j} is arc-consistent by (ii), according to Lemma 7, we know that $\delta_{v_{n-i},v_j} = \bigcap_{x\in F_i} \left(\delta_{v_{n-i},x} \circ \delta_{x,v_j}\right)$ is tree-preserving. Similarly, since

$$\delta^{-1}_{v_{n-i},v_j} = \bigcap_{x\in F_i} \left(\delta_{v_{n-i},x} \circ \delta_{x,v_j}\right)^{-1} = \bigcap_{x\in F_i} \left(\delta^{-1}_{x,v_j} \circ \delta^{-1}_{v_{n-i},x}\right) = \bigcap_{x\in F_i} \left(\delta_{v_j,x} \circ \delta_{x,v_{n-i}}\right),$$

we know that the inverse of δ_{v_{n-i},v_j} is also tree-preserving. □

By the above lemma, we now prove that the result obtained for CRC constraints in [4] also applies to tree-preserving constraints.

Theorem 3 *For a tree-preserving constraint network Δ with a triangulated constraint graph G, strong PC on G is equivalent to strong PC on the completion of G in the sense that the relations computed for the constraints in G are identical.*

Proof The proof is analogous to that for CRC constraints in [4]. Suppose we have a triangulated constraint graph $G = (V, E)$ that is strongly PC. We will add to G the missing edges one by one until the graph is complete. To prove the theorem, we show that the relations of the constraints can be computed from the existing ones so that each intermediate graph, including the completed graph, is strongly PC. Let i be the largest index such that G_i is complete. Since G_{i+1} is not complete, we add one by one all missing edges $\{v_{n-i}, v_j\}$ into G_{i+1} and compute their corresponding constraints as

$$\delta_{v_{n-i},v_j} = \bigcap_{x\in F_i} \left(\delta_{v_{n-i},x} \circ \delta_{x,v_j}\right)$$

(see Fig. 8 for an illustration, where vertices in F_i are denoted in black). By Lemma 8, we know δ_{v_{n-i},v_j} and its inverse are tree-preserving and the revised graph G'_{i+1} remains triangulated. Continuing in this way, we will transform G_{i+1} into a complete graph. Applying the above procedure to G_{i+1}, and so on, until the whole graph is complete, we will have the desired result. □

Then, we show that enforcing PPC on a consistent tree-preserving constraint network transforms it into an equivalent constraint network that is backtrack-free in the following sense.

Definition 10 ([13]) A constraint network is *backtrack-free* relative to a given ordering $\prec = \langle x_1, \cdots, x_n \rangle$ if for every $i \leq n-1$, every partial solution of $\{x_1, \ldots, x_i\}$ can be consistently extended to x_{i+1}.

We now have the main result of this section.

Theorem 4 *Suppose Δ is a tree-preserving constraint network with triangulated constraint graph G. If no inconsistency is detected, then enforcing strong PPC on G transforms it into an equivalent consistent network that is backtrack-free relative to the reverse ordering of any perfect elimination ordering $\prec = \langle v_1, \cdots, v_n \rangle$ of G.*

Proof Suppose $\prec = \langle v_1, \cdots, v_n \rangle$ is a perfect elimination ordering of G. In the following, we use notations S_i, F_i and G_i that are defined in Definition 9.

Assume that no inconsistency is detected. Write Δ^* for the equivalent network obtained from enforcing strong PPC on Δ. We show that Δ^* is backtrack-free w.r.t. the ordering $\prec^{-1} = \langle v_n, v_{n-1}, ..., v_1 \rangle$. To this end, we need to show that, for any $2 \leq i \leq n$, any consistent instantiation of vertices in $S_{i-1} = \{v_n, v_{n-1}, \cdots, v_{n-i+2}\}$ can be consistently extended to v_{n-i+1}.

Suppose the above statement holds for any $2 \leq i \leq j$. We show that it holds for $i = j+1$. Because the elimination ordering is perfect, $F_j \cup \{v_{n-j}\}$ is complete. So, the restriction of Δ into $F_j \cup \{v_{n-j}\}$ is strongly PC and tree-preserving and thus, by Theorem 2, globally consistent. Therefore, any consistent instantiation to vertices in F_j could be consistently extended to v_{n-j}. Also, because there are no edges (i.e. no constraints) between v_{n-j} and vertices in $G_j \setminus F_j$, any consistent instantiation to vertices in G_j could be consistently extended to v_{n-j} such that all constraints in G_{j+1} are satisfied. In this way, we have shown that Δ^* is backtrack-free w.r.t. $\prec^{-1}$. □

According to Theorem 4, enforcing PPC is sufficient to solve tree-preserving constraint networks. In the following, we will show that *conservative dual-consistency* (CDC) [28] is equal to PPC for tree-preserving constraint networks with triangulated constraint graphs.

Given a binary constraint network Δ, $\Delta|_{v_i=a_i}$ represents the network obtained from Δ by restricting the domain of v_i to the singleton $\{a_i\}$ and $AC(\Delta|_{v_i=a_i})$ denotes the network obtained by enforcing AC on $\Delta|_{v_i=a_i}$.

Definition 11 [28] Let Δ be a binary constraint network over variable set V. Δ is called *conservative dual-consistent* (CDC) if for any $\delta_{v_iv_j} \in \Delta$ and any $\langle a_i, a_j \rangle \in \delta_{v_iv_j}$, we have $a_j \in D_j^*$ where D_j^* is the domain of v_j w.r.t. $AC(\Delta|_{v_i=a_i})$. Δ is called strongly CDC if it is both AC and CDC.

Proposition 7 *Strong partial path-consistency is equivalent to strong conservative dual-consistency for tree-preserving constraint networks with triangulated constraint graphs.*

Proof Let Δ be a tree-preserving constraint network over variable set V. Suppose that the constraint graph $G = (V, E)$ of Δ is triangulated. If Δ is strongly CDC, then Δ is also strongly PPC, because conservative dual-consistency is a stronger consistency condition than partial path-consistency [30]. Now, suppose that Δ is strongly PPC, we show that Δ is also strongly CDC. We first obtain a new network Δ' by adding all the missing edges to G to make it complete. Constraints of newly added edges are all set to be universal. Then we enforce strong PC on Δ'. Now, Δ' is strongly PC and Δ' is equivalent to Δ. By Theorem 2, Δ' is globally consistent. Let $\delta_{v_iv_j}$ be an arbitrary constraint of Δ. By Theorem 3, $\delta_{v_iv_j}$ is also a constraint of Δ'. Then, for any tuple $\langle a_i, a_j \rangle \in \delta_{v_iv_j}$, it can be extended to a solution

ψ of Δ' which is also a solution of Δ. Therefore, we have $a_j \in D_j^*$ where D_j^* is the domain of v_j w.r.t. $AC(\Delta|_{v_i=a_i})$, and we know that Δ is CDC. Because Δ is also AC, it is strongly CDC. □

Consequently, we can adopt efficient strong CDC enforcing algorithms, such as sCDC1 [30], to enforce PPC for tree-preserving constraint networks.

Finally, we give Algorithm 1 below to find solutions for tree-preserving constraint networks.

Algorithm 1: Solving tree-preserving constraint network using PPC.

input : A tree-preserving constraint network Δ.
output: A solution or inconsistency.

```
1  Triangulate the constraint graph G(Δ);
2  Find a perfect elimination order of G(Δ), {v_1, v_2, ··· , v_n};
3  Enforce PPC on G(Δ);
4  if no inconsistency is detected then
5  |  Choose values a_n and a_{n-1} for v_n and v_{n-1} respectively s.t. (a_n, a_{n-1}) satisfies δ_{n,n-1};
6  |  for i ← n − 2 to 1 do
7  |  |  S = ⋂_{v_k ∈ F_{n-i}} δ_{ki}(v_k);
8  |  |  Pick a value a_i from S for v_i;
9  |  end
10 end
11 else return inconsistency;
```

We first explain how Algorithm 1 works, and then prove its correctness and analyse its time complexity in Theorem 5. Lines 1-3 are self-explanatory. Line 5 instantiates $G_2 = \{v_n, v_{n-1}\}$, and then Lines 6-9 consistently instantiate G_3 to G_n in a sequential way. Line 8 extends the consistent instantiation of G_i to G_{i+1} by finding a consistent value for v_{n-i}.

Take the constraint graph in Fig. 6 as an example. The algorithm first assigns consistent values a_5 and a_4 to the variables v_5 and v_4 of $G_2 = \{v_5, v_4\}$ respectively, and then extends the instantiation of G_2 to G_3 by finding a consistent value for v_3, i.e. to find a value a_3 for vertex v_3 such that $(a_i, a_3) \in \delta_{v_i,v_3}$ for all $i \in F_2$. To achieve this, Algorithm 1 computes the intersection of $\delta_{v_i,v_3}(a_i)$ for all $i \in F_2$. By Theorem 3, the intersection S is nonempty. Then Algorithm 1 picks any value from S for v_3. Similarly, the algorithm extends the instantiation of G_3 to G_4, and then extends the instantiation of G_4 to G_5.

Theorem 5 *Algorithm 1 is correct and its time complexity is $O(n(e+f) + \alpha(e+f)d^3)$, where α is the maximum degree of vertices in $G(\Delta)$, the constraint graph of the input tree-preserving constraint network, f is the number of added edges to make $G(\Delta)$ triangulated and d is the maximal domain size.*

Proof The correctness of Algorithm 1 follows directly from Theorem 3. Now, we analyse time complexity of the algorithm. Finding a minimum triangulation for $G(\Delta)$ in Line 1 could be done in $O(n(e+f))$[24], where f is the number of added edges. In Line 2, a perfect elimination ordering for the minimum triangulation of G can be found in $O(n+e+f)$ [24]. Also, enforcing PPC in Line 3 can be done in $O(\alpha(e+f)d^3)$[4], and Lines 6-9 take time $O(\alpha nd)$. Therefore, the overall time complexity of the algorithm is $O(n(e+f) + \alpha(e+f)d^3)$. □

In the next section, we consider a particular application of tree-preserving constraints.

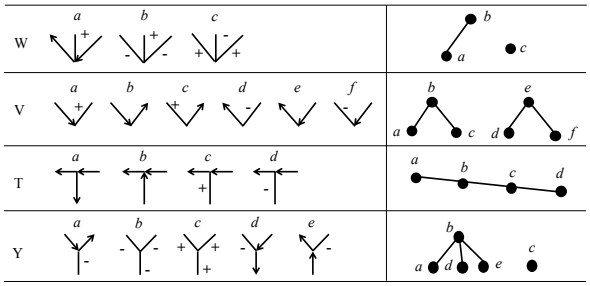

Fig. 9 Possible labelled line configurations of a junction in a picture and their corresponding forest structures.

5 The Scene Labelling Problem

The scene labelling problem [19] is a classification problem where every edge in a line-drawing picture has to be associated with a label describing it. The scene labelling problem is NP-complete in general and this is true even in the case of the trihedral scenes, i.e. scenes where no four planes share a point [23]. Several tractable subclasses of scene labelling problem have been identified (cf. [9,22]).

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

'$+$' The edge is *convex*, *i.e.*, the edge can be touched by a ball;
'$-$' The edge is *concave*, *i.e.*, the edge cannot be touched by a ball;
'$\rightarrow$' 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 [7,19]. A vertex projects into a 'V', 'W', 'Y' or 'T'-junction in the picture (each of these junction-types may appear with an arbitrary 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. 9.

In this section, we show that (i) every instance of the trihedral scene labelling problem can be modelled by a tree convex constraint network over forest domains; (ii) a large subclass of the trihedral scene labelling problem can be modelled by tree-preserving 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 variables are the possible configurations as shown in Fig. 9. The constraints between variables are simply that, if two variables share an edge, then the edge must be labelled the same at both ends.

Proposition 8 *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 can be found in the online appendix[1]. Because '—' must be labelled by an arrow from right to left and '|' can be labelled by any labels, the T-junctions decompose into unary constraints. For this reason, we do not consider T-junctions in line drawing pictures. □

[1] https://www.researchgate.net/publication/301815699_Appendix

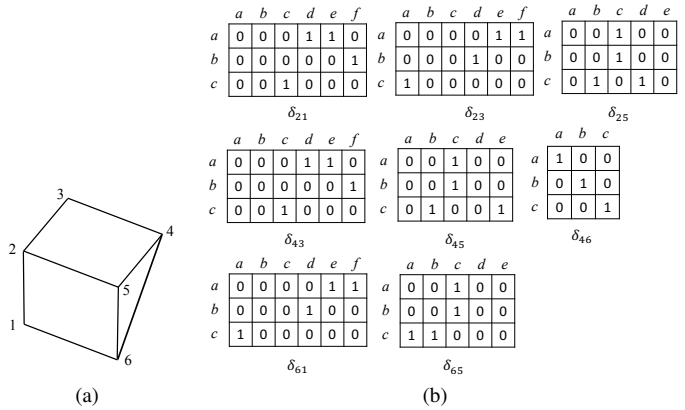

Fig. 10 (a) A line drawing. (b) Constraints for the drawing. All of them are tree-preserving constraints.

As a consequence, the consistency of any constraint network whose relations are taken from these 29 relations can be decided by enforcing strong path-consistency. Moreover, because 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.*

A scene labelling instance and its corresponding constraint network are shown in Fig. 10; the network is tree-preserving but neither chain-preserving nor CRC. Consider the line drawing on the left of Fig. 10 and the constraints for the drawing listed on the right. One can easily verify that all constraints are tree-preserving w.r.t. the forest structures listed in Fig. 9, but, for example, δ_{21} is not chain-preserving w.r.t. the forest structures illustrated in Fig. 9 and δ_{25} is not CRC.

In the following section we give another method for proving the tractability of the class of tree-preserving constraints.

6 Algebraic Closure Properties of Tree-Preserving Constraints

We have shown that strong path-consistency ensures global consistency for the classes of chain-, path-, and tree-preserving constraints and thus identified three tractable classes of binary relations. Our approach follows the research line initiated by Dechter [12] and continued in e.g. [15,35,36,38], which are based on the idea of achieving global consistency by enforcing local consistency.

An alternative approach to the study of tractable classes of relations focuses on certain algebraic closure properties of constraints [5,16,21]. In this section, we will show that, under mild restrictions, a relation is tree-preserving *if and only if* it satisfies some algebraic closure property.

We first recall some basic notions introduced in [5] and [20]. A relation (or an operation) is called *one-sorted* if it is defined over a single domain and *multi-sorted* otherwise.

Definition 12 Suppose $f^D : D^r \to D$ is a one-sorted operation on D. We say f^D is *near-unanimity* if

$$(\forall a, b \in D_i) \quad f^D(b, a, \ldots, a) = f^D(a, b, a, \ldots, a) = \ldots = f^D(a, \ldots, a, b) = a. \quad (3)$$

A ternary ($r = 3$) near-unanimity operation is called a *majority* operation.

Definition 13 Given a collection of finite sets $\mathcal{D} = \{D_1, \ldots, D_n\}$, an *$r$-ary multi-sorted operation* f is a collection of one-sorted operations $\{f^{D_1}, \ldots, f^{D_n}\}$, where $f^{D_i} : D_i^r \to D_i$. Let $\delta \subseteq D_{i_1} \times D_{i_2} \times \ldots \times D_{i_m}$ be a multi-sorted relation over $\mathcal{D}$. δ is *closed under* f if for any r m-tuples $\langle d_{11}, d_{21}, \ldots, d_{m1}\rangle$, $\langle d_{12}, d_{22}, \ldots, d_{m2}\rangle$, ..., $\langle d_{1r}, d_{2r}, \ldots, d_{mr}\rangle$ in δ, we have

$$\langle f^{D_{i_1}}(d_{11}, d_{12}, \ldots, d_{1r}), \ldots, f^{D_{i_m}}(d_{m1}, d_{m2}, \ldots, d_{mr})\rangle \in \delta$$

Definition 14 Let δ be an m-ary relation over domain $\mathcal{D} = \{D_1, \ldots, D_n\}$. We say δ is *r-decomposable* if, for any m-tuples t and any $I = (i_1, \ldots, i_k)$ (a list of indices chosen from $\{1, \ldots, m\}$) with $k \leq r$, we have $t \in \delta$ if $\pi_I(t) \in \pi_I(\delta)$, where $\pi_I(t) = \langle t[i_1], ..., t[i_k]\rangle$ and $\pi_I(\delta) = \{\pi_I(t') \mid t' \in \delta\}$.

Definition 15 For a set of relations Γ, we write Γ^+ for the set of all relations which can be obtained from Γ by using some sequence of *Cartesian product* (for $\delta_1, \delta_2 \in \Gamma, \delta_1 \times \delta_2 = \{\langle t_1, t_2\rangle \mid t_1 \in \delta_1, t_2 \in \delta_2\}$), *equality selection* (for $\delta \in \Gamma, \sigma_{i=j}(\delta) = \{t \in \delta \mid t[i] = t[j]\}$), and *projection* (for $\delta \in \Gamma, \pi_{i_1,\cdots,i_k}(\delta) = \{\langle t[i_1], \cdots, t[i_k]\rangle \mid t \in \delta\}$).

In the following, we denote by C_Γ the set of constraint networks all relations of which are taken from Γ. We have the following extension of [20, Theorem 3.5] from the one-sorted case to the multi-sorted case. The proof is similar to the one-sorted case and thus omitted.

Theorem 6 *Suppose Γ is a set of multi-sorted relations over a collection of finite sets $\mathcal{D} = \{D_1, \ldots, D_n\}$. For any $r \geq 3$, the following conditions are equivalent:*

1. *There exists an r-ary near unanimity operation f^{D_i} on D_i for each $1 \leq i \leq n$ such that every relation δ in Γ is closed under the multi-sorted operation $f = (f^{D_1}, \ldots, f^{D_n})$.*
2. *Every δ in Γ^+ is $(r-1)$-decomposable.*
3. *For every constraint network $\Delta \in C_\Gamma$, establishing strong r-consistency ensures global consistency.*

For each tree domain, we introduce a natural majority operation.

Definition 16 Let T_x be a nonempty tree domain for a variable x. We define a majority operation m_x as:

$$(\forall a, b, c \in T_x) \qquad m_x(a, b, c) = \pi_{a,b} \cap \pi_{b,c} \cap \pi_{a,c}, \quad (4)$$

where a, b, c are not necessarily distinct and $\pi_{u,v}$ denotes the unique path from u to v in T_x. We call m_x the *standard* majority operation on T_x.

Even for one-sorted relations over a tree domain T, the class of tree-preserving relations on T is not comparable to the class of relations that are closed under the standard majority operation on T (see Fig. 11 for an illustration).

The following lemma gives two important properties of relations closed under standard majority operations.

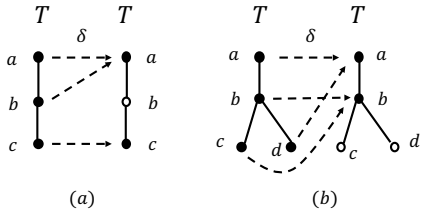

Fig. 11 (a) δ is a relation from tree domain T to T, and $\delta = \{(a,a),(b,a),(c,c)\}$. δ is closed under the standard majority operation on T, but it is not tree-preserving w.r.t. T. (b) δ is a relation from tree domain T to T, and $\delta = \{(a,a),(b,b),(c,b),(d,a)\}$. δ is tree-preserving w.r.t. T, but it is not closed under the standard majority operation on T.

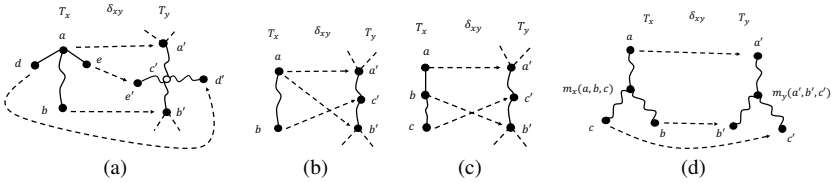

Fig. 12 Illustration of proof of proposition 9. Broken arrow lines indicate images of the node and empty circles indicates empty support nodes.

Lemma 9 *Let T_x and T_y be two nonempty tree domains and m_x and m_y their standard majority operations. Suppose $\delta \subseteq T_x \times T_y$ is a nonempty relation that is closed under $\{m_x, m_y\}$. Then*

- *δ^{-1}, the inverse of δ, is closed under $\{m_y, m_x\}$;*
- *if $a', b' \in T_y$ are the only supported nodes in the path from a' to b', then $\delta^{-1}(M_{a',b'}) = \varnothing$, where $M_{a',b'}$ is the trunk between a', b' in T_y.*

Proof The first result follows directly from the definition.

To prove the second result, suppose $\langle a, a'\rangle$ and $\langle b, b'\rangle \in \delta$ and no other nodes in $\pi_{a',b'}$ are supported (see Fig. 12 (a) for an illustration). If there exist $c' \in M_{a',b'}$ and $c \in T_x$ such that $\langle c, c'\rangle \in \delta$, then we have $\langle m_x(a,b,c), m_y(a',b',c')\rangle \in \delta$. It is clear that $m_y(a',b',c')$ is a node in $\pi_{a',b'}$ which is different from a' and b'. This is a contradiction and hence the statement is correct. □

Let $T'_y = T_y \ominus M_{a',b'}$ and m'_y the standard majority operation on tree T'_y. Based on Lemma 9, it is easy to see that, when restricted to T'_y, δ and its inverse are also closed under $\{m_x, m'_y\}$. If we continue contracting and revising T_y and T_x in this way, then, in finite steps, we will reach a state in which every node is supported. Write the revised tree domains in this state as T^*_x and T^*_y and let m^*_x and m^*_y be their corresponding standard majority operations. Then, when restricted to T^*_x and T^*_y, δ and δ^{-1} are closed under $\{m^*_x, m^*_y\}$. This suggests that it is reasonable to consider relations that are arc-consistent.

Based upon this observation, we have the following characterisation.

Proposition 9 *Let T_x and T_y be two nonempty tree domains and m_x and m_y their standard majority operations. Suppose $\delta \subseteq T_x \times T_y$ is a nonempty relation such that both δ and its inverse, δ^{-1}, are arc-consistent. Then δ is closed under $\{m_x, m_y\}$ iff both δ and δ^{-1} are tree-preserving w.r.t. T_x and T_y.*

Proof Suppose δ is closed under $\{m_x, m_y\}$. We first prove that δ_{xy} is tree convex. Suppose not. Then we have $a \in T_x$, $a', b', c' \in T_y$ such that $\langle a, a'\rangle, \langle a, b'\rangle \in \delta$, $c' \in \pi_{a',b'}$ but $\langle a, c'\rangle \notin \delta$ (see Fig. 12 (b) for an illustration). Because δ^{-1} is arc-consistent, there exists $c \in T_x$ such that $\langle c, c'\rangle \in \delta$. Consider the three tuples $\langle a, a'\rangle, \langle a, b'\rangle, \langle c, c'\rangle$ in δ. We have $\langle m_x(a, a, c), m_y(a', b', c')\rangle \in \delta$ as δ is closed under $\{m_x, m_y\}$. Since $m_x(a, a, c) = a$ and $m_y(a', b', c') = c'$, we have $\langle a, c'\rangle \in \delta$, which is a contradiction. Therefore, δ_{xy} is tree convex. Next, we prove that δ is consecutive. Suppose not. Then there exist two neighbouring nodes $a, b \in T_x$ such that $\delta(a) \cup \delta(b)$ is not a subtree of T_y. This means that there are $a' \in \delta(a)$, $b' \in \delta(b)$, and $c' \in T_y$ such that c' is in $\pi_{a'b'}$ but not in either $\delta(a)$ or $\delta(b)$. Because δ^{-1} is arc-consistent, there exists c in T_x such that $\langle c, c'\rangle \in \delta$ (see Fig. 12 (c) for an illustration). Consider the three tuples $\langle a, a'\rangle, \langle b, b'\rangle, \langle c, c'\rangle$ in δ. We have $\langle m_x(a, b, c), m_y(a', b', c')\rangle \in \delta_{xy}$. Because a is a neighbour of b, $m_x(a, b, c) = \pi_{ab} \cap \pi_{bc} \cap \pi_{ac}$ is either a or b. Therefore, we have either $\langle a, c'\rangle \in \delta$ or $\langle b, c'\rangle \in \delta$, which is a contradiction. Therefore, δ_{xy} is arc-consistent and consecutive tree convex and, hence, tree-preserving. Similarly, δ^{-1} is tree-preserving since it is also closed under $\{m_x, m_y\}$.

On the other hand, suppose both δ and δ^{-1} are tree-preserving. We prove that δ is closed under $\{m_x, m_y\}$. Take three arbitrary tuples $\langle a, a'\rangle, \langle b, b'\rangle, \langle c, c'\rangle$ from δ. We need to prove $\langle m_x(a, b, c), m_y(a', b', c')\rangle \in \delta$. For convenience, we denote m, m' for $m_x(a, b, c)$ and $m_y(a', b', c')$ respectively. Because δ is tree-preserving, from $a', b' \in \delta(\pi_{a,b})$, we know $\pi_{a',b'}$ is contained in $\delta(\pi_{a,b})$. In particular, $m' \in \delta(\pi_{a,b})$. Similarly, we also have $m' \in \delta(\pi_{a,c})$ and $m' \in \delta(\pi_{b,c})$. Note that $\pi_{a,b} = \pi_{a,m} \cup \pi_{b,m}$, $\pi_{b,c} = \pi_{b,m} \cup \pi_{c,m}$, and $\pi_{a,c} = \pi_{a,m} \cup \pi_{c,m}$. We know m' belongs to at least two of $\delta(\pi_{a,m})$, $\delta(\pi_{b,m})$, and $\delta(\pi_{c,m})$. Suppose $m' \notin \delta(\pi_{c,m})$. Then $\langle m, m'\rangle \notin \delta$ and $m' \in \delta(\pi_{a,m})$ and $m' \in \delta(\pi_{b,m})$. There are $a_1 \in \pi_{a,m}$ and $b_1 \in \pi_{b,m}$ such that $\langle a_1, m'\rangle \in \delta$ and $\langle b_1, m'\rangle \in \delta$. Since δ^{-1} is tree-preserving, $\delta^{-1}(m')$ is a subtree of T_x. From $\pi_{a_1,b_1} \subseteq \delta^{-1}(m')$ and $m \in \pi_{a_1,b_1}$, we know $m \in \delta^{-1}(m')$, i.e. $\langle m, m'\rangle \in \delta$, which is a contradiction. Therefore, the assumption that $m' \notin \delta(\pi_{c,m})$ is incorrect. This implies that m' belongs to each of $\delta(\pi_{a,m})$, $\delta(\pi_{b,m})$, and $\delta(\pi_{c,m})$. We therefore have $a_1 \in \pi_{a,m}$, $b_1 \in \pi_{b,m}$, and $c_1 \in \pi_{c,m}$ such that $\langle a_1, m'\rangle$, $\langle b_1, m'\rangle$, and $\langle c_1, m'\rangle$ are all in δ. Let t be the subtree spanned by a_1, b_1, c_1 in T_x. Then t is contained in $\delta^{-1}(m')$ since δ^{-1} is tree-preserving. From $m \in t$, we have the desired result that $\langle m, m'\rangle \in \delta$. □

Remark 2 Proposition 9 establishes the connection between tree-preserving constraints and those binary constraints that are closed under the standard majority operation. Using this result, it is natural to extend the definition of tree-preserving constraints to non-binary tree-preserving constraints. For a non-binary relation R, assuming that it is arc-consistent in each variable, we may call R a tree-preserving constraint if it is closed under the standard majority operation induced by the relevant tree domains.

Using Proposition 9, we now give an alternative proof for Theorem 2.

Proof of Theorem 2: Let $\Delta = \{x_i \delta_{ij} x_j \mid 1 \leq i, j \leq n\}$ be a tree-preserving constraint network. We note that if no inconsistency is detected after enforcing arc-consistency, then Δ remains tree-preserving (see Proposition 2). Without loss of generality, we suppose Δ is arc-consistent. Write T_i for the tree-domain of variable x_i. Let Γ be the set of binary relations over $\mathcal{D} = \{T_i \mid 1 \leq i \leq n\}$ that are closed under the multi-sorted operation $f = (m_{x_i} \mid 1 \leq i \leq n)$, where each m_{x_i} is the standard majority operation over T_i. By Proposition 9, every relation δ_{ij} in Δ is closed under $\{m_{x_i}, m_{x_j}\}$. That is, each δ_{ij} is

a relation in Γ and thus Δ is an instance of C_Γ. By Theorem 6, we have the result that establishing strong path-consistency ensures global consistency for Δ. □

Proposition 9 considers only tree domains. We can also generalise it to forest domains.

Definition 17 Given a forest domain F_x with trees $\{t_1, ..., t_n\}$, the *standard majority operation* m_x for F_x is defined as:

$$m_x(a,b,c) = \begin{cases} \pi_{ab} \cap \pi_{bc} \cap \pi_{ac} & \text{if } a,b,c \in t_i \ (1 \leq i \leq n), \\ a & \text{if } a,b \in t_i,\ c \in t_j; \text{ or } a,c \in t_i,\ b \in t_j \ (i \neq j), \\ b & \text{if } a \in t_i,\ b,c \in t_j \ (i \neq j), \\ a & \text{if } a \in t_i,\ b \in t_j,\ c \in t_k \ (i \neq j \neq k). \end{cases}$$

We note that the order of a, b, c matters. That is, for example, $m_x(a,b,c)$ may not be the same as $m_x(b,c,a)$.

Proposition 10 *If δ and its inverse δ^{-1} are both arc-consistent and tree-preserving over forest domains F_x and F_y, then δ and δ^{-1} are closed under $\{m_x, m_y\}$ and $\{m_y, m_x\}$ respectively.*

Proof Because δ and δ^{-1} are arc-consistent and tree-preserving, there is a bijection between trees in $F_x = \{t_1, t_2, ..., t_n\}$ and trees in $F_y = \{t'_1, t'_2, ..., t'_n\}$ such that $\delta(t_i) = t'_i$ and $\delta^{-1}(t'_i) = t_i$ for every $1 \leq i \leq n$. For any $\langle a, a' \rangle, \langle b, b' \rangle, \langle c, c' \rangle \in \delta$, consider the following cases separately.

(1) Suppose $a, b, c \in t_i$ for some $t_i \in F_x$. Because δ and δ^{-1} are tree-preserving, a', b', c' are all in t'_i. Then, by Proposition 9, $\langle m_x(a,b,c), m_y(a',b',c') \rangle \in \delta$.
(2) Suppose $a, b \in t_i$ and $c \in t_j$, or $a, c \in t_i$ and $b \in t_j$ for some $i \neq j$. We have $a', b' \in t'_i$ and $c' \in t'_j$, or $a', c' \in t'_i$ and $b' \in t'_j$. Thus $\langle m_x(a,b,c), m_y(a',b',c') \rangle = \langle a, a' \rangle \in \delta$.
(3) Suppose $a \in t_i$ and $b, c \in t_j$ for some $i \neq j$. We have $a' \in t'_i$ and $b', c' \in t'_j$. Thus $\langle m_x(a,b,c), m_y(a',b',c') \rangle = \langle b, b' \rangle \in \delta$.
(4) Suppose $a \in t_i, b \in t_j$ and $c \in t_k$ for some pairwise different i, j, k. We have $a' \in t'_i, b' \in t'_j$ and $c' \in t'_k$. Thus $\langle m_x(a,b,c), m_y(a',b',c') \rangle = \langle a, a' \rangle \in \delta$. □

Remark 3 For the 39 different relations in the trihedral scene labelling problem, 29 of them are tree-preserving. These 29 relations are all closed under the standard majority operations defined above for forest domains. As we have expected, the other ten relations are not closed under the standard majority operations.

7 Experiments

In this section, we report experimental evaluations of local consistency enforcing algorithms for *sparse* tree-preserving constraint networks.

As we have shown in Section 4, enforcing strong PPC is sufficient to solve tree-preserving constraint networks. Although enforcing PPC should be generally faster than enforcing PC for sparse constraint networks [4], for practical interests, we conduct experimental comparisons between PPC algorithms and their counterparts, PC algorithms, for solving sparse tree-preserving constraint networks. We consider two competitive strong PC algorithms, sDC1 [29,30] and PC2001 [3], for comparisons. It is worthwhile to note that sDC1 is a strong *dual consistency* (DC) enforcing algorithm and DC has been shown to be equal to PC for binary constraint networks [29,30]. Just like PC, DC considers every distinct pair

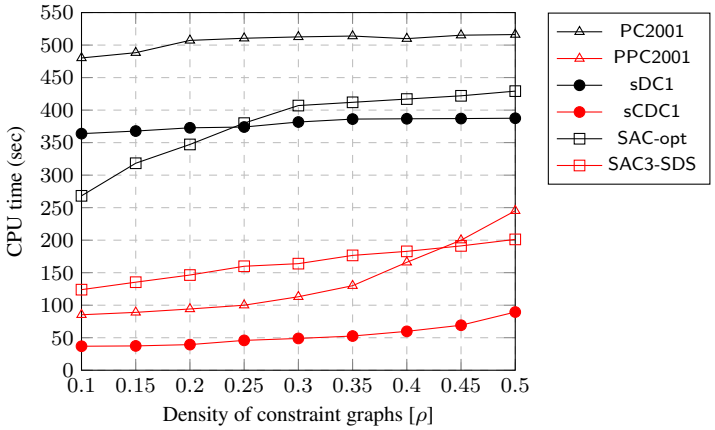

Fig. 13 Performance evaluation of different local consistency algorithms for solving tree-preserving constraint networks with different densities. We set $n = 100$, $d = 30$ and $l = 0.5$.

of variables of binary constraint networks. However, *conservative* DC (CDC) only considers a pair $\{v_i, v_j\}$ if there is a constraint $\delta_{v_i v_j}$ imposed on it. As Proposition 7 suggests strong CDC is equivalent to strong PPC for tree-preserving constraint networks with triangulated constraint graphs, we adopt the efficient strong CDC enforcing algorithm sCDC1 to enforce strong PPC on such networks. We also devise another strong PPC algorithm, called PPC2001, upon the algorithm PC2001. We do it by modifying PC2001 to enforce PC on a triangulation instead of the completion of the input constraint graph.

By Proposition 9, the class of tree-preserving constraints is closed under the standard majority operation, and thus tree-preserving constraint networks can also be solved by enforcing *singleton arc-consistency* (SAC) (see e.g., [6]). We also include two state-of-the-art SAC algorithms, SAC3-SDS [1] and SAC-opt [2], for comparisons. However, it is worthwhile to note that it is unknown whether enforcing SAC would enable backtrack-free search for tree-preserving constraint networks.

By Proposition 7, a random tree-preserving constraint network can be generated as follows:

(1) For every domain D_x of the network, generate a random tree T_x for with vertex set D_x.
(2) For each pair of variables x and y, generate an arc-consistent relation $\delta_{xy} \subseteq D_x \times D_y$ s.t. δ_{xy} is closed under $\{m_x, m_y\}$.

Four parameters are used to generate a random tree-preserving network: (1) n - the number of variables, (2) d - the size of the domains, (3) ρ - the density of the constraint graph (i.e. the ratio of non-universal constraints to n^2), (4) l - the looseness of the constraints (i.e. the ratio of the number of allowed tuples to d^2).

All constraints are represented as Boolean matrices. Experimentation was carried out on a computer with an Inter Core i5 processor with a frequency of 2.9 GHz per CPU core, 8 GB of RAM, and the MAC OSX. The experimental platform is Eclipse with JDK 8.

Experimental results are presented in Fig. 13, where each test is averaged over 20 instances. We can observe that lower densities of constraint graphs do not benefit the performances of sCDC1 and PC2001 much. On the other hand, SAC3-SDS, SAC-opt, sCDC1 and PPC2001 are all exploiting the sparsity of constraint networks. In particular, they all perform better when constraint networks are sparser. The PPC algorithms sCDC1 and

PPC2001 can outperform their counterparts, the PC algorithms sCD1 and PC2001, by up to a factor of 7 and 3.5 respectively. Moreover, sCDC1 beats all the other considered algorithms.

SAC3-SDS performs reasonably well for sparse tree-preserving constraint networks and is comparable to PPC2001. It also outperforms SAC-opt roughly by a factor of 2, but its performance is about twice worse than that of sCDC1. Unexpectedly, sDC1 is comparable to SAC-opt for sparse tree-preserving constraint networks, because SAC is a weaker consistency condition than PC and thus SAC algorithms are usually expected to be more efficient than PC algorithms.

8 Conclusion

The study of tractable subclasses of constraint satisfaction problems is one of the most important research problems in artificial intelligence. In this paper, we studied three tractable subclasses of tree convex constraints, which are generalisations of the well-known row convex constraint. We proved that enforcing strong path-consistency decides the consistency of a tree-preserving constraint network and, if no inconsistency is detected, transforms the network into a globally consistent constraint network. Actually, we proved this by two methods. The first method directly proved that enforcing strong path-consistency transforms a tree-preserving constraint network into a path-consistent tree-preserving network, while the second method relied on the characterisation of tree-preserving constraints by closure under majority operations. Since every arc-consistent chain- or path-preserving constraint is a tree-preserving constraint, we got a tractable subclass of CSPs that is genuinely larger than the subclass of CRC constraints. We further showed that PPC algorithms can be applied to solve tree-preserving constraint networks in a backtrack-free style, which is more efficient than using a standard path-consistency algorithm. As an application, we proved that every relation used in the trihedral scene labelling problem can be modelled by a tree convex constraint, and, among these different relations (39 in total), 29 are tree-preserving constraints. This means that a large tractable subclass of the NP-hard trihedral scene labelling problem can be solved by the techniques discussed in this paper. As the class of tree-preserving constraints has the Helly property as stated in Lemma 1, it is not difficult to show that the deterministic distributed algorithm for solving CRC constraints, proposed in [25] and called DΔCRC, can also be adapted to solving tree-preserving constraints.

Acknowledgments

This work was partially supported by an ARC Discovery Project (No. DP120104159). The work of Sanjiang Li was also supported by NSFC (No. 11671244).

References

1. Bessiere, C., Cardon, S., Debruyne, R., Lecoutre, C.: Efficient algorithms for singleton arc consistency. Constraints **16**(1), 25–53 (2011)
2. Bessiere, C., Debruyne, R.: Theoretical analysis of singleton arc consistency and its extensions. Artificial Intelligence **172**(1), 29–41 (2008)
3. Bessière, C., Régin, J., Yap, R.H.C., Zhang, Y.: An optimal coarse-grained arc consistency algorithm. Artificial Intelligence **165**(2), 165–185 (2005)

4. Bliek, C., Sam-Haroud, D.: Path consistency on triangulated constraint graphs. In: IJCAI, pp. 456–461 (1999)
5. Bulatov, A.A., Jeavons, P.: An algebraic approach to multi-sorted constraints. In: CP, pp. 183–198 (2003)
6. Chen, H., Dalmau, V., Grußien, B.: Arc consistency and friends. Journal of Logic and Computation **23**(1), 87–108 (2011)
7. Clowes, M.B.: On seeing things. Artificial Intelligence **2**(1), 79–116 (1971)
8. Conitzer, V., Derryberry, J., Sandholm, T.: Combinatorial auctions with structured item graphs. In: AAAI, pp. 212–218 (2004)
9. Cooper, M.C.: Linear-time algorithms for testing the realisability of line drawings of curved objects. Artificial Intelligence **108**(1), 31–67 (1999)
10. Cooper, M.C., Jeavons, P.G., Salamon, A.Z.: Hybrid tractable csps which generalize tree structure. In: ECAI, pp. 530–534 (2008)
11. Cooper, M.C., Mouelhi, A.E., Terrioux, C., Zanuttini, B.: On broken triangles. In: CP, pp. 9–24 (2014)
12. Dechter, R.: From local to global consistency. Artificial Intelligence **55**(1), 87–107 (1992)
13. Dechter, R.: Constraint processing. Morgan Kaufmann (2003)
14. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artificial Intelligence **49**(1-3), 61–95 (1991)
15. Deville, Y., Barette, O., Hentenryck, P.V.: Constraint satisfaction over connected row convex constraints. Artificial Intelligence **109**(1-2), 243–271 (1999)
16. 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)
17. Freuder, E.C.: Synthesizing constraint expressions. Communications of the ACM **21**(11), 958–966 (1978)
18. Freuder, E.C.: A sufficient condition for backtrack-free search. Journal of the ACM **29**(1), 24–32 (1982)
19. Huffman, D.A.: Impossible objects as nonsense sentences. Machine Intelligence **6**(1), 295–323 (1971)
20. Jeavons, P., Cohen, D.A., Cooper, M.C.: Constraints, consistency and closure. Artificial Intelligence **101**(1-2), 251–265 (1998)
21. Jeavons, P., Cohen, D.A., Gyssens, M.: Closure properties of constraints. Journal of the ACM **44**(4), 527–548 (1997)
22. Kirousis, L.M.: Effectively labelling planar projections of polyhedra. IEEE Transactions on Pattern Analysis and Machine Intelligence **12**(2), 123–130 (1990)
23. Kirousis, L.M., Papadimitriou, C.H.: The complexity of recognizing polyhedral scenes. In: FOCS, pp. 175–185 (1985)
24. Kjærulff, U.: Triangulation of graphs – algorithms giving small total state space. Tech. rep. (1990)
25. Kong, S., Lee, J.H., Li, S.: A distributed deterministic algorithm for reasoning with connected row-convex constraints. In: AAMAS, to appear (2017)
26. Kong, S., Li, S., Li, Y., Long, Z.: On tree-preserving constraints. In: CP, pp. 244–261 (2015)
27. Kumar, T.K.S.: Simple randomized algorithms for tractable row and tree convex constraints. In: AAAI, pp. 74–79 (2006)
28. Lecoutre, C., Cardon, S., Vion, J.: Conservative dual consistency. In: AAAI, pp. 237–242 (2007)
29. Lecoutre, C., Cardon, S., Vion, J.: Path consistency by dual consistency. In: CP, pp. 438–452 (2007)
30. Lecoutre, C., Cardon, S., Vion, J.: Second-order consistencies. Journal of Artificial Intelligence Research **40**, 175–219 (2011)
31. Li, S., Liu, W., Wang, S.: Qualitative constraint satisfaction problems: An extended framework with landmarks. Artificial Intelligence **201**, 32–58 (2013)
32. Mackworth, A.K.: Consistency in networks of relations. Artificial Intelligence **8**(1), 99–118 (1977)
33. Maruyama, H.: Structural disambiguation with constraint propagation. In: ACL, pp. 31–38 (1990)
34. Montanari, U.: Networks of constraints: fundamental properties and applications to picture processing. Information Sciences **7**, 95–132 (1974)
35. 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)
36. Zhang, Y., Freuder, E.C.: Tractable tree convex constraint networks. In: AAAI, pp. 197–203 (2004)
37. Zhang, Y., Freuder, E.C.: Properties of tree convex constraints. Artificial Intelligence **172**(12-13), 1605–1612 (2008)
38. Zhang, Y., Yap, R.H.C.: Consistency and set intersection. In: IJCAI, pp. 263–270 (2003)

Appendix: Proofs

Lemma 5 *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 no inconsistency is detected.*

Proof The proof of this result heavily uses Lemma 13 and Proposition 2.

First, we consider how trunks propagate in the network. Starting from the trunk $M_{a,b}$ in T_x, we get, as in Lemma 13, a unique trunk M_{a_y,b_y} in T_y for each $y \neq x$ in V if $\delta_{xy}(a) \cup \delta_{xy}(b)$ is not connected in T_y. Furthermore, each trunk M_{a_y,b_y} (in T_y) will also propagate in the network, obtaining a (possibly new) trunk $M_{a_{yz},b_{yz}}$ in T_z for each $z \neq y$. Continuing this way, we stop until no new trunks are generated. Since there are finitely many different trunks in each tree domain, the process will stop in a finite number of steps. Write $\mathcal{T}_y$ for the set of trunks obtained for variable y. We note that nodes in these trunks have to be deleted from the corresponding tree domain to maintain arc-consistency.

We next amalgamate these trunks in each $\mathcal{T}_y$. By Lemma 11, the union of two connected trunks can be the whole tree, a branch, or a larger trunk. Similarly, we can prove that if a trunk and a branch are connected, then their union is a branch, a trunk, or the whole tree. If the union of all trunks in a $\mathcal{T}_y$ is the whole tree, viz. T_y, then the network is inconsistent. In the following, we assume that this is not the case. This implies that the trunks in $\mathcal{T}_y$ can be merged into a set of pairwise disconnected maximal trunks and maximal branches. Let t_y be the subtree of T_y obtained after removing all these maximal branches. We now restrict the constraint network to subtrees t_y ($y \in V$) and enforce arc-consistency. By Corollary 1 and Proposition 2, we get a new arc-consistent tree-preserving network Δ' over smaller tree domains, say T'_y ($y \in V$), if no inconsistency is detected.

Consider the original trunk $M_{a,b}$ in T_x. If $M_{a,b} \cap T'_x = \varnothing$ or $T'_x \subseteq M_{a,b}$, then we need do nothing as the network is either tree-preserving or trivially inconsistent after contracting $M_{a,b}$. If $M_{a,b} \cap T'_x$ is a branch of T'_x, then we use Corollary 1 again and transform Δ' into a new arc-consistent and tree-preserving network if no inconsistency is detected. If neither of the above happens, then $M_{a,b} \cap T'_x$ is a trunk in T'_x. We repeat the above process again and again until no new branches are generated.

From now on, we suppose that no branches are obtained by merging trunks in any $\mathcal{T}_y$. Furthermore, for each variable y, we denote by $\mathcal{MT}_y$ the set of maximal trunks of T_y after amalgamation. We contract the maximal trunks in $\mathcal{MT}_y$ one by one and write T^*_y for the contracted tree.

For any two variables $y \neq z$, we show that δ_{yz}, when restricted to T^*_y and T^*_z, remains tree-preserving. Suppose $\mathcal{MT}_y = \{M_{a_1,b_1}, ..., M_{a_k,b_k}\}$ and assume, without loss of generality, that the first $m \leq k$ trunks satisfy the precondition of Lemma 13, i.e. $\delta_{yz}(a_i) \cup \delta_{yz}(b_i)$ is disconnected for $1 \leq i \leq m$. By Lemma 13, there exists a unique trunk $M_{a'_i,b'_i}$ in T_z such that $a'_i \in \delta_{yz}(a_i)$, $b'_i \in \delta_{yz}(b_i)$ and $\delta_{zy}(M_{a'_i,b'_i}) \subseteq M_{a_i,b_i}$ for each $1 \leq i \leq m$.

Let $\mathcal{MN}_y = \{M_{a_1,b_1}, ..., M_{a_m,b_m}\}$ and $\mathcal{N}_z = \{M_{a'_1,b'_1}, ..., M_{a'_m,b'_m}\}$. Note that trunks in $\mathcal{N}_z$ are not necessarily maximal. Let $M_{a''_i,b''_i}$ be the maximal trunk in $\mathcal{MT}_z$ which contains $M_{a'_i,b'_i}$.[2] We write $\mathcal{MN}_z = \{M_{a''_1,b''_1}, ..., M_{a''_m,b''_m}\}$. We now show how to contract these trunks so that we get T^*_y and T^*_z while preserving the tree-preserving property.

First, we contract all maximal trunks in $\mathcal{MT}_y$ that are not in $\mathcal{MN}_y$. Because the images of the two nodes a_i, b_i under δ_{yz} are connected in T_z, the constraint δ_{yz} remains tree-preserving after the contraction. Let T'_y be the resultant tree domain of y.

Second, we contract all maximal trunks in $\mathcal{MN}_y$ from T'_y and contract all corresponding trunks in $\mathcal{N}_z$ from T_z. Clearly, the resultant tree domain of y is exactly T^*_y. Denote by T'_z the resultant tree domain of z. By Lemma 13, δ_{yz} is tree-preserving when restricted to T^*_y and T'_z. Note that, after the contraction of $M_{a'_i,b'_i}$ from T_z, $M_{a''_i,b''_i}$ becomes a trunk in

[2] Since no branches can be obtained by merging trunks in $\mathcal{T}_z$, we know that $M_{a'_i,b'_i}$ is contained in a maximal trunk.

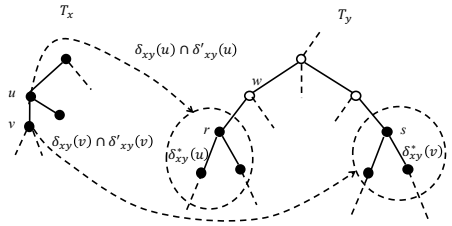

Fig. 14 Illustration of proof of Lemma 7.

T'_z. We then contract all these trunks together with all other maximal trunks in $\mathcal{MT}_z$ from T'_z. The resultant tree domain is exactly T^*_z.

We show that δ_{yz} is still tree-preserving when restricted to T^*_y and T^*_z. Given any subtree t of T^*_y, because δ_{yz} is tree-preserving w.r.t. T^*_y and T'_z, we know $t' \equiv \delta_{yz}(t)$ is a nonempty subtree of T'_z. By Lemma 12 we know t' is a (possibly empty) subtree of T^*_z. This proves that δ_{yz} is still tree-preserving when restricted to T^*_y and T^*_z.

Because the arbitrariness of y, z above, we know every δ_{uw}, in particular δ_{zy}, is tree-preserving w.r.t. T^*_u and T^*_w. Note that these constraints are not necessarily arc-consistent. By Proposition 2 again, we transform these constraints into arc-consistent constraints while remaining tree-preserving. In summary, we know that, when restricted to $T_x \ominus M_{a,b}$ and enforcing arc-consistency, Δ remains tree-preserving if no inconsistency is detected. □

Lemma 6 *Assume δ_{xy} and δ'_{xy} are two arc-consistent and tree-preserving constraints w.r.t. trees T_x and T_y. Let $\delta^*_{xy} = \delta_{xy} \cap \delta'_{xy}$. Let $W = \{w \in T_x \mid \delta^*_{xy}(w) \neq \varnothing\}$ be the set of supported values of δ^*_{xy}. Suppose $u \in T_x$ and $u \notin W$. Then there exist at most two values w_1, w_2 in W s.t. no value in W other than w_i is on the path $\pi_{w_i,u}$ for $i = 1, 2$.*

Proof Suppose w_1, w_2, w_3 are three supported values of δ^*_{xy} in T_x s.t. no value in W other than w_j is on the path $\pi_{w_j,u}$ for $1 \leq j \leq 3$. Take $w'_i \in \delta^*_{xy}(w_i)$ $(i = 1, 2, 3)$. Let $\{u_1\} = \pi_{w_1,w_2} \cap \pi_{w_1,w_3} \cap \pi_{w_2,w_3}$ and $\{u'_1\} = \pi_{w'_1,w'_2} \cap \pi_{w'_1,w'_3} \cap \pi_{w'_2,w'_3}$. By the choice of w_1, w_2, w_3, they cannot be on a same path. In particular, u_1 is different from w_1, w_2, w_3. Furthermore, since u_1 is on π_{w_1,w_2}, it must be on either π_{u,w_1} or π_{u,w_2}. In either case, we have u_1 is not in W.

Because δ_{xy} is tree-preserving, from $w_i, w_j \in \delta_{yx}(\pi_{w'_i,w'_j})$, we know π_{w_i,w_j} is contained in $\delta_{yx}(\pi_{w'_i,w'_j})$ for any $1 \leq i \neq j \leq 3$. In particular, u_1 is in $\delta_{yx}(\pi_{w'_i,w'_j})$. In other words, $\delta_{xy}(u_1) \cap \pi_{w'_i,w'_j} \neq \varnothing$. By Lemma 1, we know $\delta_{xy}(u_1) \cap \pi_{w'_1,w'_2} \cap \pi_{w'_1,w'_3} \cap \pi_{w'_2,w'_3} \neq \varnothing$. Because $\{u'_1\} = \pi_{w'_1,w'_2} \cap \pi_{w'_1,w'_3} \cap \pi_{w'_2,w'_3}$, we know $u'_1 \in \delta_{xy}(u_1)$. Analogously, we have $u'_1 \in \delta'_{xy}(u_1)$ and, hence, u_1 is a value in W. This is a contradiction. Therefore, there exist at most two values w_1, w_2 in W s.t. no value in W other than w_j is on the path $\pi_{w_j,u}$ for $i = 1, 2$. □

Lemma 7 *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. $\delta^*_{xy}(u) \neq \varnothing$, $\delta^*_{xy}(v) \neq \varnothing$, and $\delta^*_{xy}(u) \cup \delta^*_{xy}(v)$ is disconnected in T_y. Then there exist unique $r \in \delta^*_{xy}(u)$ 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 (unique) $r \in T_r$, $s \in T_s$ s.t. $\pi_{r,s} \cap (T_r \cup T_s) = \{r, s\}$ (see Fig. 14 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 unsupported under δ^*_{yx} either.

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

The following simple properties are used to assist the proofs of Lemmas 5-7.

Lemma 10 *Suppose $M_{a,b}$ and u are, respectively, a trunk and a node of tree T. Let $t_a = \{w \in T \mid a \in \pi_{w,b}\}$ and $t_b = \{w \in T \mid b \in \pi_{w,a}\}$. Then*

(i) $a \in t_a$, $b \in t_b$, $a, b \notin M_{a,b}$. Moreover, t_a and t_b are subtrees of T separated by $M_{a,b}$ and $\{t_a, t_b, M_{a,b}\}$ is a partition of T.

(ii) $u \notin M_{a,b}$ iff $a \in \pi_{u,b}$ or $b \in \pi_{u,a}$; if $u \in M_{a,b}$, then $\pi_{u,a} \subseteq M_{a,b} \cup \{a\}$ and $\pi_{u,b} \subseteq M_{a,b} \cup \{b\}$.

Proof Take a as the root of T. Let a_1 be the child of a that is on the path $\pi_{a,b}$. Then t_a is the subtree obtained by removing the subtree rooted at a_1, and t_b is the subtree rooted at b. The results are then clear. □

The next lemma considers the union of two connected trucks.

Lemma 11 *Suppose $M_{a,b}$ and $M_{c,d}$ are two trunks of tree $T = (V, E)$. Then $M_{a,b} \cup M_{c,d}$ is connected if and only if either (i) $M_{a,b} \cap M_{c,d} \neq \varnothing$ or (ii) there exist $x \in \{a, b\}$, $y \in \{c, d\}$ s.t. $\{x, y\}$ is an edge in T and $x \in M_{c,d}, y \in M_{a,b}$. Moreover, if $M_{a,b} \cup M_{c,d}$ is connected, then it is T, or a trunk or branch of T.*

Proof If either (i) or (ii) holds, then clearly $M_{a,b} \cup M_{c,d}$ is connected. Suppose $M_{a,b} \cup M_{c,d}$ is connected but $M_{a,b} \cap M_{c,d} = \varnothing$, we show (ii) holds. Because $M_{a,b} \cup M_{c,d}$ is connected, there exist $u \in M_{a,b}$, $v \in M_{c,d}$ such that $\{u, v\}$ is an edge in T. In addition, because $M_{a,b} \cap M_{c,d} = \varnothing$, we have $v \notin M_{a,b}$, $u \notin M_{c,d}$. Then by Lemma 10, we know $a \in \pi_{v,b}$ or $b \in \pi_{v,a}$, and $c \in \pi_{u,d}$ or $d \in \pi_{u,c}$. Without loss of generality, suppose $a \in \pi_{v,b}$ and $c \in \pi_{u,d}$. From $a \in \pi_{v,b}$ and $u \in M_{a,b}$, we have $u \in \pi_{v,b}$. Because u is a neighbour of v, this is possible only if $v = a$ and $u \in \pi_{a,b}$. Analogously, we also have $u = c$ and $v \in \pi_{c,d}$. Therefore, we have $\{a, c\} \in E$, $c \in M_{a,b}$ and $a \in M_{c,d}$. Note that in this case $M_{a,b} \cup M_{c,d} = M_{b,d}$ is another trunk.

We next suppose $M_{a,b} \cap M_{c,d} \neq \varnothing$ and $c, d \notin M_{a,b}$. We show that $a \in M_{c,d}$ iff $b \in M_{c,d}$. Suppose otherwise $a \in M_{c,d}$ but $b \notin M_{c,d}$. Then by $b \notin M_{c,d}$ and Lemma 10 we have either $c \in \pi_{b,d}$ or $d \in \pi_{b,c}$. In either case, by $a \in M_{c,d}$ and Lemma 10, we have $c \in \pi_{a,b}$ or $d \in \pi_{a,b}$, which contradicts the assumption that $c, d \notin M_{a,b}$. Thus, we have $a \in M_{c,d}$ iff $b \in M_{c,d}$.

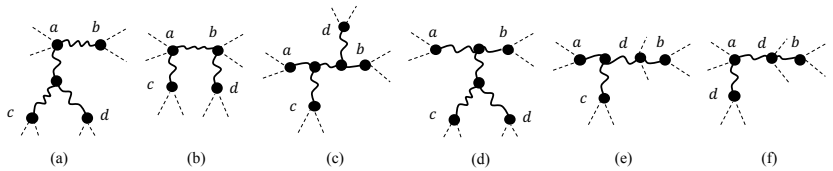

Fig. 15 Possible configurations of two connected trunks $M_{a,b}$ and $M_{c,d}$, where $M_{a,b} \cup M_{c,d}$ is the whole tree in (c) and (d), a trunk in (a), (b) and (f), and a branch in (e).

We now consider the following possible cases (symmetric cases are omitted):

(1) Suppose $c, d \notin M_{a,b}$ and $a, b \notin M_{c,d}$. By Lemma 10, we have $a \in \pi_{c,b}$ or $b \in \pi_{c,a}$, $a \in \pi_{d,b}$ or $b \in \pi_{d,a}$, $c \in \pi_{a,d}$ or $d \in \pi_{a,c}$, and $c \in \pi_{b,d}$ or $d \in \pi_{b,c}$. Since $M_{a,b} \cap M_{c,d} \neq \varnothing$, it is straightforward to show that $\{a, b\} = \{c, d\}$ and $M_{a,b} = M_{c,d}$.

(2) Suppose $c, d \notin M_{a,b}$ and $a, b \in M_{c,d}$. Because $c, d \notin M_{c,d}$, we know a, b, c, d are pairwise different. Moreover, we have $a \in \pi_{c,b}$ or $b \in \pi_{c,a}$, and $a \in \pi_{d,b}$ or $b \in \pi_{d,a}$. We can prove that $M_{a,b} \cup M_{c,d} = M_{c,d}$ is a trunk. There are two possible configurations (see Fig. 15 (a) and (b)).

(3) Suppose $c, d \in M_{a,b}$ and $a, b \in M_{c,d}$. Then, a, b, c, d are pairwise different. We can prove that $M_{a,b} \cup M_{c,d}$ is the whole tree. There are two possible configurations (see Fig. 15(c) and (d)).

(4) Suppose $c, d \in M_{a,b}$, $a \in M_{c,d}$, but $b \notin M_{c,d}$. Then a, b, c, d are pairwise different. We can prove that $M_{a,b} \cup M_{c,d}$ is a branch of b. There is only one possible graph (see Fig. 15(e)).

(5) Suppose $c \in M_{a,b}, d \notin M_{a,b}$, $a \in M_{c,d}$, and $b \notin M_{c,d}$. Then $c \neq a, b$ and $a \neq c, d$, but it is possible that $b = d$. In this case, we can prove that $a, c \in \pi_{d,b}$, $a \in \pi_{d,c}$, and $c \in \pi_{a,b}$ (see Fig. 15(f)). Moreover, we have $M_{a,b} \cup M_{c,d} = M_{d,b}$ is a trunk. □

The next lemma considers what happens to a subtree when we contract a trunk from the tree domain.

Lemma 12 *Suppose $M_{a,b}$ and t are, respectively, a trunk and a subtree of tree T. If t is not contained in $M_{a,b}$, then t, when restricted to $T \ominus M_{a,b}$, is also a subtree of $T_x \ominus M_{a,b}$.*

Proof Let $t_a = \{u \in T \mid a \in \pi_{u,b}\}$ and $t_b = \{u \in T \mid b \in \pi_{u,a}\}$. By Lemma 10, $a \in t_a$, $b \in t_b$, t_a and t_b are two subtrees separated by $M_{a,b}$, and $\{t_a, t_b, M_{a,b}\}$ is a partition of T.

If $t \subseteq M_{a,b}$, then all nodes in t are deleted after the contraction of T by $M_{a,b}$; if $t \cap M_{a,b} = \varnothing$, then no nodes in t are deleted after the contraction of T by $M_{a,b}$; if $t \not\subseteq M_{a,b}$ and $t \cap M_{a,b} \neq \varnothing$, there are three subcases. First, if $t \cap t_a \neq \varnothing$ and $t \cap t_b \neq \varnothing$, then both a, b are in t. After contraction, t is the union of two subtrees $t \cap t_a$ and $t \cap t_b$, which are connected by the new edge $\{a, b\}$. Hence, t is still a subtree. Second, if $t \cap t_a \neq \varnothing$ but $t \cap t_b = \varnothing$, then $a \in t$ but $b \notin t$. After contraction, t will be replaced by $t \cap t_a$. Third, if $t \cap t_a = \varnothing$ and $t \cap t_b \neq \varnothing$, then, after contraction, t will be replaced by $t \cap t_b$. □

Given a tree-preserving constraint δ_{xy} 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. We now consider how to modify T_y so that δ_{xy} remains tree-preserving after contracting trunk $M_{a,b}$ from T_x.

Lemma 13 *Suppose δ_{xy} and δ_{yx} are arc-consistent and tree-preserving w.r.t. tree domains T_x and T_y. Let a, b be two nodes in T_x s.t. $\delta_{xy}(a) \cup \delta_{xy}(b)$ is not connected in T_y. Then there exist unique $r, s \in T_y$ s.t. $r \in \delta_{xy}(a)$, $s \in \delta_{xy}(b)$, and $\delta_{yx}(M_{r,s}) \subseteq M_{a,b}$. Moreover, δ_{xy} and δ_{yx} are tree-preserving when restricted to $T_x \ominus M_{a,b}$ and $T_y \ominus M_{r,s}$.*

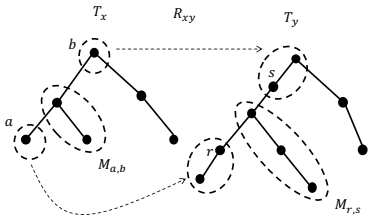

Fig. 16 Illustration of proof of Lemma 13.

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 $\{\pi_{r',s'} : r' \in \delta_{xy}(a), s' \in \delta_{xy}(b)\}$ (see Fig. 16 for an illustration). 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, for every node c in $T_x \ominus M_{a,b}$, since c is supported by a node in $T_y \ominus M_{r,s}$, we know that $\delta_{xy}(c)$ is a nonempty subtree in T_y. By Lemma 12 and $\delta_{xy}(c) \cap M_{r,s} = \varnothing$, we know $\delta_{xy}(c) \cap (T_y \ominus M_{r,s})$ is also a nonempty subtree in $T_y \ominus M_{r,s}$. This shows that δ_{xy} is tree-preserving when restricted to $T_x \ominus M_{a,b}$ and $T_y \ominus M_{r,s}$. On the other hand, for any subtree t in $T_y \ominus M_{r,s}$, w.l.o.g., assume that $r, s \in t$. Then $\delta_{yx}(t) = \delta_{yx}(t \cap t_r) \cup \delta_{yx}(t \cap t_s)$, where t_r and t_s are the two connected components of T_y separated by $M_{r,s}$. Because both $\delta_{yx}(t \cap t_r)$ and $\delta_{yx}(t \cap t_s)$ are subtrees in T_x and, hence, subtrees in $T_x \ominus M_{a,b}$ by Lemma 12. By $a \in \delta_{yx}(t \cap t_r)$ and $b \in \delta_{yx}(t \cap t_s)$, we know $\delta_{yx}(t)$ is a subtree of $T_y \ominus M_{r,s}$. This shows that δ_{yx} is tree-preserving when restricted to $T_x \ominus M_{a,b}$ and $T_y \ominus M_{r,s}$. □

It is possible that there is a node $v \in T_y \ominus M_{r,s}$ s.t. $\delta_{yx}(v) \subseteq M_{a,b}$, but the image of $T_x \ominus M_{a,b}$ under the restricted δ_{xy} is a subtree of $T_y \ominus M_{r,s}$.