Simple Randomized Algorithms for Tractable Row and Tree Convex Constraints

T. K. Satish Kumar
Computer Science Division
University of California, Berkeley
tksk@eecs.berkeley.edu

Abstract

We identify tractable classes of constraints based on the following simple property of a constraint: "At every infeasible point, there exist two directions such that with respect to any other feasible point, moving along at least one of these two directions decreases a certain distance metric to it". We show that connected row convex (CRC) constraints, arc-consistent consecutive tree convex (ACCTC) constraints, etc fit this characterization, and are therefore amenable to extremely simple polynomial-time randomized algorithms—the complexities of which are shown to be much less than that of the corresponding (known) deterministic algorithms and the (generic) lower bounds for establishing path-consistency. On a related note, we also provide a simple polynomial-time deterministic algorithm for finding tree embeddings of variable domains (if they exist) for establishing tree convexity in path-consistent networks.

Introduction

While the task of solving constraint satisfaction problems (CSPs), in general, is NP-hard, much work has been done on identifying tractable subclasses. Broadly, these subclasses have resulted from restrictions imposed on: (1) the *topology* of the associated constraint network (Dechter 1992), (2) the structure of the *constraints* themselves (see (Van Beek and Dechter 1995), (Jeavons *et al* 1998), (Deville *et al* 1999) and (Bulatov *et al* 2000)), or (3) a combination of both (Dechter and Pearl 1991). While the notions of minimum induced-width and hypergraph acyclicity play a key role in characterizing the complexity of solving a given CSP by looking only at the topology of its associated constraint network (Dechter 1992), the notions of row and tree convexity, among others, have been identified in the context of exploiting the structure of the constraints themselves (see (Van Beek and Dechter 1995) and (Zhang and Freuder 2004)). Row and tree convex constraints generalize other types of constraints like monotone and functional constraints (Van Hentenryck *et al* 1992), and together with relational path-consistency, ensure the global consistency of a constraint network.

More specifically, if a binary constraint network is path-consistent, and all of the binary relations can be made row or tree convex by finding suitable domain orderings for the variables, then the network is globally consistent—enabling us to find a solution in a backtrack-free manner. However, neither row convex constraints nor tree convex constraints by themselves constitute tractable languages since further conditions are necessary to ensure that the additional constraints resulting from enforcing path-consistency also remain row or tree convex respectively. Connected row convexity (CRC) is a further restriction on row convexity that ensures the closure over composition, intersection and transposition—the basic operations of path-consistency algorithms—hence making the language of CRC constraints tractable (Deville *et al* 1999). Similarly, tractable classes of tree convex constraints satisfying the properties of local chain convexity and union closure are identified in (Zhang and Freuder 2004). An algebraic theory of tractable constraint languages is developed in (Jeavons *et al* 1998), etc.

In this paper, we identify tractable classes of constraints based on the following simple property of a constraint: "At every infeasible point, there exist two directions such that with respect to any other feasible point, moving along at least one of these two directions decreases a certain distance metric to it". We show that CRC constraints, arc-consistent consecutive tree convex (ACCTC) constraints, etc fit this characterization, and are therefore amenable to extremely simple polynomial-time randomized algorithms—the complexities of which are shown to be much less than that of the corresponding (known) deterministic algorithms and the (generic) lower bounds for establishing path-consistency. Further, to the best of our knowledge, ACCTC constraints were not previously identified to be tractable. Finally, on a related note, we also provide a simple polynomial-time deterministic algorithm for finding tree embeddings of variable domains (if they exist) for establishing tree convexity in path-consistent networks.

Preliminaries and Definitions

A CSP is defined by a triplet $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{X} = \{X_1, X_2 \ldots X_N\}$ is a set of *variables*, and $C = \{C_1, C_2 \ldots C_M\}$ is a set of *constraints* between subsets of them. Each variable X_i is associated with a discrete-valued *domain* $D_i \in \mathcal{D}$, and each constraint C_i is a pair $\langle S_i, R_i \rangle$ defined on a subset of variables $S_i \subseteq \mathcal{X}$, called the *scope* of C_i. $R_i \subseteq D_{S_i}$ ($D_{S_i} = \times_{X_j \in S_i} D_j$) denotes all compatible tuples of D_{S_i} allowed by the constraint. A *solution* to a

ALGORITHM: PATH-CONSISTENCY
INPUT: A binary constraint network $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$.
OUTPUT: A path-consistent network.
(1) Repeat until no constraint is changed:
  (a) For $k = 1, 2 \ldots N$:
    (i) For $i, j = 1, 2 \ldots N$:
      (A) $R_{ij} = R_{ij} \cap \Pi_{ij}(R_{ik} \bowtie D_k \bowtie R_{kj})$.
END ALGORITHM

Figure 1: Basic algorithm for enforcing *path-consistency* in a binary constraint network. Π indicates the *projection* operation, and $\bowtie$ indicates the *join* operation (similar to that in database theory).

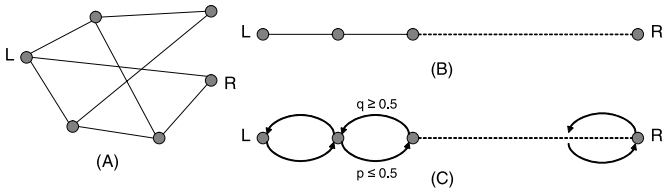

Figure 2: Shows 3 scenarios in which random walks are performed. In an undirected graph ((A) and (B)), for any 2 nodes L and R, $T(R, L) + T(L, R)$ is related to the "resistance" between them. In (C) ($p \leq q$ at every node), $T(R, L)$ is less than that in (B) because of an increased "attraction" towards L at every node.

CSP is an assignment of values to all the variables from their respective domains such that all the constraints are satisfied.

A network of binary constraints is *path-consistent* if and only if for all variables X_i, X_j and X_k, and for every instantiation of X_i and X_j that satisfies the direct relation R_{ij}, there exists an instantiation of X_k such that R_{ik} and R_{kj} are also satisfied. Conceptually, path-consistency enforcing algorithms work by iteratively "tightening" the binary constraints as shown in Figure 1. The best known algorithm that implements this procedure exploiting low-level consistency maintenance is presented in (Mohr and Henderson 1986), and has a running time complexity of $O(N^3K^3)$ (K is the size of the largest domain). This algorithm is optimal, since even verifying path-consistency has the same lower bound.

When binary relations are represented as matrices, path-consistency algorithms employ the three basic operations of composition, intersection and transposition. The (0,1)-matrix representation of a relation R_{ij} (denoted $M_{R_{ij}}$) between variables X_i and X_j consists of $|D_i|$ rows and $|D_j|$ columns when orderings on the domains of X_i and X_j are imposed. The '1's and '0's in the matrix respectively indicate *allowed* and *disallowed* tuples.

Random Walks and Expected Arrival Times

We will provide a quick overview of random walks, and some theoretical properties associated with them. Figure 2(A) shows an undirected graph (unit weights assumed on all edges). A random walk on such a graph involves starting at a particular node, and at any stage, randomly moving to one of the neighboring positions of the current position. A property associated with such random walks on undirected graphs is that if we denote the expected time of arrival at some node (say L) starting at a particular node (say R) by

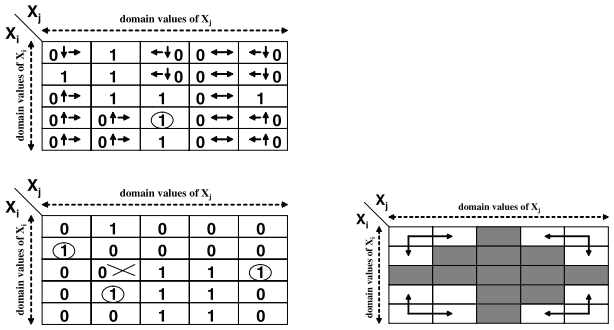

Figure 3: Illustrates the notion of a smooth constraint (left side). The right side illustrates the general pattern of the required pair of directions for '0's in a CRC constraint (shaded areas indicate '1's).

$T(R, L)$, then $T(R, L) + T(L, R)$ is $O(m\mathcal{H}(L, R))$; m is the number of edges, and $\mathcal{H}(L, R)$ is the "resistance" between L and R, when the unit weights on edges are interpreted as electrical resistance values (Doyle and Snell 1984).

Figure 2(B) shows a particular case of the one in Figure 2(A), in which the nodes in the graph are connected in a linear fashion—i.e. the probabilities of moving to the left or to the right from a particular node are equal (except at the end-points). In this scenario, it is easy to note that by symmetry, $T(L, R) = T(R, L)$. Further, using the property of random walks stated above, if there are n nodes in the graph, then both $T(L, R)$ and $T(R, L)$ are $O(n^2)$.

Figure 2(C) shows a slightly modified version of that in Figure 2(B), where the graph is directed, although it is still linear. Moreover, there are weights associated with edges, which are interpreted as probabilities in the random walk; and the weight on $\langle s, s_{left} \rangle$ is, in general, not equal to that on $\langle s, s_{right} \rangle$. Here, s is some node in the graph, and s_{left} and s_{right} are respectively the nodes occurring immediately to the left and right of it. However, we are guaranteed that the probability of moving to the left at any node is greater than that of moving to the right (i.e. $p \leq q$). Given this scenario, it is easy to see that the expected time of arrival at the left end-point (L), starting at the right end-point (R), is also $O(n^2)$ (if there are n nodes in all). Informally, this is because at every node, there is an increased "attraction" to the left compared to that in Figure 2(B); and the expected arrival time can only be less than that in the latter.

Tractable Row Convex Constraints

A binary relation R_{ij} represented as a (0,1)-matrix, is *row convex* if and only if, in each row, all of the '1's are consecutive. If there exists an ordering of the domains of $X_1, X_2 \ldots X_N$ in a path-consistent network of binary constraints, such that all the relations can be made row convex, then the network is globally consistent (Van Beek and Dechter 1995). A globally consistent network has the property that a solution can be found in a backtrack-free manner. A (0,1)-matrix is *connected row convex* if, after removing empty rows and columns, it is row convex and *connected* (i.e. the positions of the '1's in any two consecu-

tive rows intersect, or are consecutive).[1] A binary relation R_{ij} constitutes a CRC constraint if both $M_{R_{ij}}$ and $M^T_{R_{ij}}$ are connected row convex. Unlike row convex constraints, CRC constraints are closed under composition, intersection and transposition—hence establishing that path-consistency guarantees global consistency (Deville *et al* 1999). An instantiation of the path-consistency algorithm that further exploits CRC constraints has a running time of $O(N^3K^2)$ and a space complexity of $O(N^2K)$ (Deville *et al* 1999).

In (Kumar 2005a), the theory of random walks (previous section) is shown to be useful in identifying tractable classes of constraints referred to as *smooth constraints*. (See also (Papadimitriou 1991) for randomized algorithms for 2-SAT.) A binary constraint between X_i and X_j (under an ordering of their domain values) is said to be *smooth* if the following property (**P1**) is true of its matrix representation: "At every '0', there exist two directions such that with respect to every other '1' in the matrix, moving along at least one of these two directions (by 1 unit) decreases the L_1-distance to it".

Figure 3 (left side) illustrates the notion of smoothness. One of the constraints shown is a smooth binary constraint, and the other is not. In the first case (top), every '0' is marked with two directions—indicating that no matter which '1' we have in mind (encircled in figure), moving along at least one of these two directions decreases the L_1-distance between the '0' and the '1'. In other words, if we randomly choose to move in one of these two directions, we decrease the L_1-distance between the '1' and the '0' by at least 1 with a probability at least 0.5, and increase it by at most 1 with a probability at most 0.5. In the second case (bottom), no such pair of directions exists for one of the '0's (marked with a cross), and it is therefore not smooth.

Simple randomized algorithms for solving smooth constraints are presented in (Kumar 2005a). Roughly, the idea is to start out with an initial random assignment to all the variables from their respective domains, and use the violated constraints in every iteration to guide the search for the satisfying assignment A^* (if it exists). In particular, in every iteration, a violated constraint is chosen, and the rank of the assignment of one of the participating variables (according to the domain orderings) is either increased or decreased. Since we know that the true assignment A^* satisfies all constraints, and therefore the chosen one too, randomly moving along one of the two directions associated with the '0' corresponding to the current assignment A will reduce the L_1-distance to A^* with a probability ≥ 0.5. Much like the random walk in Figure 2(C), therefore, we can bound the convergence time to A^* by a quantity that is only quadratic in the maximum L_1-distance between any two complete assignments. (We also note that the L_1-distance between two complete assignments A_1 and A_2 is at most NK, and 0 if and only if $A_1 = A_2$.) See (Kumar 2005a) for more details and further reductions in the time/space complexities.

Quite interestingly, CRC constraints are smooth (Kumar 2005a), and are amenable to simple polynomial-time randomized algorithms (with time/space complexities better than that of the corresponding deterministic algorithms).

[1] Empty rows and columns are those that contain only '0's.

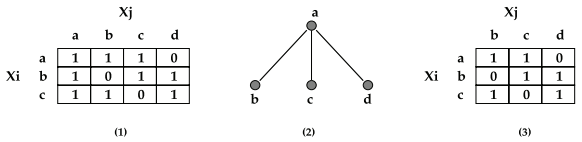

Figure 4: Illustrates why deleting a value may destroy tree convexity (borrowed from (Zhang and Freuder 2004)). (1) shows a constraint between variables X_i (domain constituting the rows) and X_j (domain constituting the columns). (2) shows a tree structure on D_j with respect to which the constraint is tree convex. (3) shows that if the value 'a' is deleted from D_j, then there does not exist any tree structure on D_j to establish tree convexity.

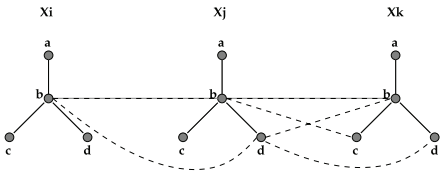

Figure 5: Illustrates why composition might destroy local chain convexity in consecutive tree convex constraint networks (borrowed from (Zhang and Freuder 2004)). The image of $b \in D_i$ is $\{b, d\}$ under $C(X_i, X_j)$, and the images of $b \in D_j$ and $d \in D_j$ are respectively $\{b, c\}$ and $\{b, d\}$ in D_k. Composing the constraints $C(X_i, X_j)$ and $C(X_j, X_k)$ therefore yields the image $\{b, c, d\}$ in D_k for $b \in D_i$, destroying the property of local chain convexity.

Figure 3 (right side) roughly illustrates the general pattern of the required pair of directions for '0's in a CRC constraint.

Tractable Tree Convex Constraints

We will first review tree convexity, and recount some of its associated results. We will then present an extremely simple randomized algorithm for solving ACCTC constraints—thereby establishing their tractability. We argue that ACCTC constraints are not only richer (more general) than those identified in (Zhang and Freuder 2004), but the time/space complexities of solving the latter (by virtue of the algorithm presented for the former) are also significantly improved.

A binary constraint network is *tree convex* if we can construct a tree (T_i) for the domain of each variable (X_i) so that for any constraint $(C(X_i, X_j))$, no matter what value one variable (say X_j) takes, all the values allowed for the other variable (X_i) form a subtree of the constructed tree (T_i). It is known that, like a row convex constraint network, a tree convex constraint network is globally consistent if it is path-consistent. However, much like the problem associated with row convexity, if a tree convex network is not path-consistent, enforcing path-consistency on it may destroy tree convexity—and hence the resulting network may not be globally consistent. Analogous to the properties of CRC constraints, (Zhang and Freuder 2004) study the impact of the intersection and composition operations on tree convex constraints and identify additional restrictions on them that ensure their tractability.

We will now briefly recount some of the definitions and results presented in (Zhang and Freuder 2004). This will help in explicating (and comparing) our own ideas.

Definition 1: The *image* of a value $u \in D_i$ under $C(X_i, X_j)$ is the set of all values in D_j that are consistent with u. The *image* of a subset of D_i under $C(X_i, X_j)$ is the union of the images of its constituent values.

Definition 2: A tree convex constraint $C(X_i, X_j)$ is *locally chain convex* with respect to a tree T_j on D_j if and only if the image of every value in D_i is a subchain of the tree—i.e. a subtree where all nodes have at most one child. A tree convex constraint network is said to be *locally chain convex* if every constraint is locally chain convex with respect to the trees on the domains of the participating variables.

The notion of local chain convexity arises in studying the impact of the intersection operation on tree convex networks. The intersection of two subtrees may be an empty set—which means that the image of a value could be empty in the intersection of two tree convex constraints. Deleting such a value could potentially destroy tree convexity (as shown in Figure 4). However, (Zhang and Freuder 2004) show that a locally chain convex constraint network remains so after the removal of any value from any domain.

Definition 3: A tree convex constraint $C(X_i, X_j)$ is *consecutive* with respect to a tree T_i on D_i if and only if the image of every subtree in T_i (under $C(X_i, X_j)$) is also a subtree in T_j. Similarly, a tree convex constraint network is said to be *consecutive* if every constraint is consecutive with respect to the trees on the domains of the participating variables.

The notion of consecutiveness arises in studying the impact of the composition operation on tree convex networks. (Zhang and Freuder 2004) show that consecutive tree convex constraints are closed under composition.

Definition 4: A locally chain convex constraint $C(X_i, X_j)$ (with respect to a tree T_j on D_j) is said to be *strictly union closed* with respect to a tree T_i on D_i if and only if the image of any subchain of T_i is a subchain of T_j. A tree convex constraint network is said to be *strictly union closed* if every constraint is strictly union closed with respect to the trees on the domains of the participating variables.

The notion of strict union closure is stronger than that of consecutiveness (i.e. the former implies the latter, but not vice versa—see (Zhang and Freuder 2004) for an example), and arises in the interest of identifying classes of tree convex constraints that are closed under both the intersection and the composition operations. While composition might destroy local chain convexity in consecutive tree convex networks (see Figure 5), it does not do so in strictly union closed tree convex networks (Zhang and Freuder 2004). (Zhang and Freuder 2004) also prove that a locally chain convex and strictly union closed tree convex constraint network can be solved in polynomial time by establishing path-consistency and doing backtrack-free search.

Definition 5: A locally chain convex constraint $C(X_i, X_j)$ (with respect to a tree T_j on D_j) is said to be *union closed* with respect to a tree T_i on D_i if and only if the image of any subchain of T_i is either a subchain of T_j, or the whole tree T_j. Similarly, a tree convex constraint network is said to be *union closed* if every constraint is union closed with respect to the trees on the domains of the participating variables.

It can be shown that even locally chain convex and union closed tree convex constraints constitute a tractable subclass of tree convex constraints, and can therefore be solved in polynomial time (Zhang and Freuder 2004).

In the rest of this section, we will make the following contributions. First, we will provide a polynomial-time algorithm for automatically constructing the required trees on the domains of the variables (if they exist) in order to achieve tree convexity for path-consistent networks. This algorithm is a simple adaptation of an algorithm presented in (Conitzer *et al* 2004) in the context of solving combinatorial auctions. Second, we will provide an extremely simple randomized algorithm for solving ACCTC constraints. For arc-consistent tree convex constraints, this class is shown to be much richer than locally chain convex and union closed constraints. Further, the same algorithm can be adapted very simply to solve general locally chain convex and union closed tree convex constraints (which may not be arc-consistent) by first establishing arc-consistency. In turn, this also means that the time/space complexities for solving them can be significantly reduced as compared to the general path-consistency enforcing algorithm of (Zhang and Freuder 2004).

Automatic Construction of Trees

As mentioned before, a tree convex constraint network is globally consistent if it is path-consistent. An important task, therefore, is to verify whether a path-consistent network can be made tree convex by automatically constructing the required tree embeddings for variable domains (if they exist). In this subsection, we will present a simple polynomial-time algorithm for doing this. This algorithm is a simple adaptation of an algorithm presented in (Conitzer *et al* 2004) in the context of solving combinatorial auctions; a corresponding algorithm for row convex constraints appears in (Van Beek and Dechter 1995) (based on (Booth and Lueker 1976) for testing the "consecutive ones" property).

Given a path-consistent network, consider the task of constructing a tree T_i (over D_i) for the variable X_i such that for any value of any other variable, the set of values in D_i that are consistent with it constitute a subtree in T_i. For every value v of every other variable X_j, we construct the set $S_{j_v} \subseteq D_i$ of values in D_i that are consistent with it. Now, given a set of such subsets $S_1, S_2 \ldots S_k$ (one for each value of every other variable), we construct a weighted undirected graph as follows. The nodes of the graph correspond to the domain values in D_i, and an undirected edge between two nodes is assigned a weight equal to the number of subsets $S_1, S_2 \ldots S_k$ in which these two nodes (corresponding domain values) occur together. The required tree T_i (if it exists) is given by the maximum weighted spanning tree on this graph (which can be computed very efficiently). A rigorous proof for this claim is presented in (Conitzer *et al* 2004).

Randomized Algorithm for ACCTC Constraints

We will now present a simple polynomial-time randomized algorithm for solving ACCTC constraints. As mentioned above, this algorithm establishes the tractability of a much richer class of tree convex constraints, and has excellent time/space complexities—also avoiding the use of complex data structures and routines otherwise required for optimally implementing path-consistency. Figure 6 presents

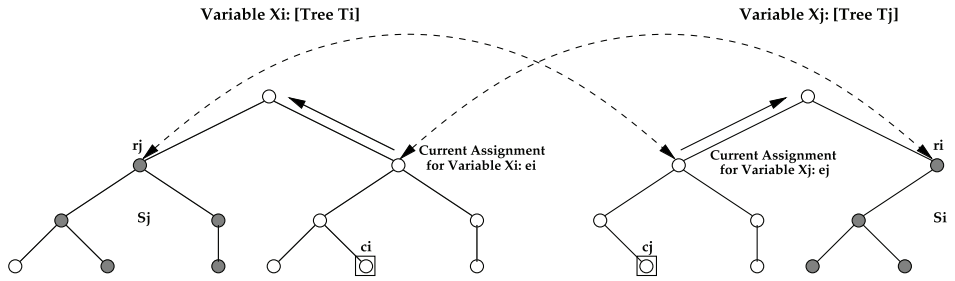

Figure 7: Illustrates the idea of performing a random walk on the tree embeddings of the domain values of variables (for solving ACCTC constraints). The current assignments to the variables X_i and X_j are e_i and e_j respectively. T_i is the tree embedding of the domain values of variable X_i, and T_j is the tree embedding of the domain values of variable X_j. The set of dark nodes (S_j) in T_i indicates the image of $X_j = e_j$, and the set of dark nodes (S_i) in T_j indicates the image of $X_i = e_i$. The nodes in square boxes (c_i and c_j) indicate some alleged consistent combination of values to the variables X_i and X_j respectively (used in the proof by contradiction).

ALGORITHM: SOLVE-ACCTC-CONSTRAINTS
INPUT: M ACCTC constraints over $\{X_1, X_2 \ldots X_N\}$.
OUTPUT: A solution to the set of constraints.
(1) Let the tree over the domain D_i of variable X_i be T_i.
(2) Start with a random assignment I to all the variables.
(3) While the current (complete) assignment A violates some constraint $C(X_i, X_j)$:
  (a) Let e_i and e_j be the current assignments to the variables X_i and X_j respectively.
  (b) Let S_j be the subtree in T_i that constitutes the image of e_j under $C(X_i, X_j)$. Let r_j be the root of this subtree.
  (c) Let e'_i be that neighbor of e_i in T_i which lies on the path between e_i and r_j.
  (d) Let S_i be the subtree in T_j that constitutes the image of e_i under $C(X_i, X_j)$. Let r_i be the root of this subtree.
  (e) Let e'_j be that neighbor of e_j in T_j which lies on the path between e_j and r_i.
  (f) Do one of the following with equal probabilities:
    (A) Set $X_i \leftarrow e'_i$.
    (B) Set $X_j \leftarrow e'_j$.
(4) RETURN: current assignment.
END ALGORITHM

Figure 6: Simple polynomial-time randomized algorithm for solving ACCTC constraints. We assume simple preprocessing steps on the constraints (similar to that in (Kumar 2005a)) so that r_i, r_j, e'_i and e'_j can be found in constant time.

this algorithm, and the following Definitions and Lemmas rigorously establish its correctness.

Definition 6: Given a tree embedding T_i for the domain D_i of each variable X_i, the *tree-based distance* between two values u and v in D_i is defined to be the length of the shortest path between u and v in T_i.

Definition 7: Given $\langle X_1 = d_{(1,i_1)}, X_2 = d_{(2,i_2)} \ldots X_N = d_{(N,i_N)} \rangle$ and $\langle X_1 = d_{(1,j_1)}, X_2 = d_{(2,j_2)} \ldots X_N = d_{(N,j_N)} \rangle$ (two complete assignments), and tree embeddings T_i for the domain D_i of each variable X_i ($1 \leq i \leq N$), the *tree-based distance* between the two complete assignments is defined to be $\mathcal{TR}(d_{(1,i_1)}, d_{(1,j_1)}, T_1) + \mathcal{TR}(d_{(2,i_2)}, d_{(2,j_2)}, T_2) \ldots \mathcal{TR}(d_{(N,i_N)}, d_{(N,j_N)}, T_N)$.
Here, $\mathcal{TR}(d_{(p,i_p)}, d_{(p,j_p)}, T_p)$ denotes the tree-based distance between $d_{(p,i_p)}$ and $d_{(p,j_p)}$ in T_p ($1 \leq p \leq N$).

We will show that an ACCTC constraint satisfies the following property (**P2**): "At every '0', there exist two directions such that with respect to every other '1', moving along at least one of these two directions (by 1 edge) decreases the tree-based distance to it". Figure 7 illustrates this claim and supports the following rigorous argument for its truth.

Consider a violated constraint $C(X_i, X_j)$. Let the tree embeddings of the domains D_i and D_j be T_i and T_j respectively. Let e_i and e_j be the current assignments for X_i and X_j respectively, and let the images of e_i and e_j in T_j and T_i (respectively) be S_i and S_j (respectively). In T_i, we move from e_i towards S_j; and in T_j, we move from e_j towards S_i; and we need to prove that at least one of them decreases the tree-based distance to any consistent combination of values c_i and c_j to X_i and X_j respectively. That is, we need to prove that both c_i and c_j cannot be in opposite directions in the respective trees. Let us assume the contrary—i.e. suppose there exist c_i and c_j that are in opposite directions with respect to the corresponding current assignments and subtrees. Consider tree T_i. The image of c_j includes c_i, and the image of e_j is S_j. Therefore, by consecutiveness, e_i must be in the image of at least one of the nodes between e_j and c_j. Conversely, this means that the image of e_i must include this node—in addition to S_i. By consecutiveness, again, this means that e_j must be in the image of e_i—contradicting the fact that $C(X_i, X_j)$ is a violated constraint under the current assignment. This proves that at least one of the prescribed directions decreases the tree-based distance to any consistent combination of values to X_i and X_j. We also note that the only reason arc-consistency is required is to be able to concretely compute the required pair of directions using the current assignment and the non-empty subtrees S_i and S_j.

Like in the case of CRC constraints, the foregoing arguments suggest an extremely simple randomized algorithm for solving ACCTC constraints (as presented in Figure 6). The idea is to start with an initial random assignment to all the variables from their respective domains, and use the violated constraints in every iteration to guide the search for the true assignment A^* (if it exists). In particular, in every iteration, a violated constraint is chosen, and the assignment of one of the participating variables is changed to move closer to the subtree of values consistent with the current assign-

ment of the other variable. Since we know that the true assignment A^* satisfies all constraints, and therefore the chosen one too, randomly moving along one of these two directions will reduce the tree-based distance to A^* with a probability ≥ 0.5. Much like the random walk in Figure 2(C), therefore, we can bound the convergence time to A^* by a quantity that is only quadratic in the maximum tree-based distance between any two complete assignments. (The tree-based distance between two complete assignments A_1 and A_2 is at most $J_1 + J_2 \ldots J_N$, and 0 if and only if $A_1 = A_2$; J_i is the diameter of the tree T_i, and in turn, is $\leq |D_i|$.) We note that (like in any Monte-Carlo algorithm), we will find a solution (if it exists) with a very high probability (using probability amplification), and will not report any solution if in fact none exist. If the algorithm does not terminate within a certain number of iterations, we can conclude that no solution exists (with a very high probability). The associated success (error) probability is over the internal coin flips of the algorithm, and not over the input instances; the algorithm works for all input instances. Further, this success (failure) probability can be amplified to a factor arbitrarily close to 1 (0) using the standard Amplification Lemma.

The expected running time of Figure 6 is $O(MN^2K^2)$: N is the number of variables, M is the number of constraints, and K is the size of the largest domain (or more precisely—the size of the largest diameter). The factor M arises due to the inner loop of the procedure (see step (3)), and can be reduced to the degree of the underlying constraint network (d) by caching the violated constraints in every iteration—making the running time equal to $O(N^2K^2d)$. The basic idea is to exploit the fact that it is sufficient for us to consider *any* violated constraint in every iteration. This is because every violated constraint is ACCTC, and gives us a chance to move closer to the solution with a probability ≥ 0.5; see (Kumar 2005a) for more details and the required data structures in the context of CRC constraints.

It is easy to note that for arc-consistent tree convex constraints, consecutiveness yields a class of tractable constraints far richer than locally chain convex and union closed constraints (see paragraphs after Definition 4). Further, the same algorithm can be adapted very simply to solve general locally chain convex and union closed tree convex constraints (that may not be arc-consistent) by first establishing arc-consistency. This is because locally chain convex and union closed tree convex constraints remain so even after the deletion of any value from any variable's domain (Zhang and Freuder 2004). This also means that the time/space complexities for solving general locally chain convex and union closed tree convex constraints are significantly reduced—to less than even the lower bounds for establishing path-consistency (otherwise used in (Zhang and Freuder 2004)).

Related and Future Work and Conclusions

We identified tractable classes of constraints based on the following simple property of a constraint: "At every infeasible point, there exist two directions such that with respect to any other feasible point, moving along at least one of these two directions decreases a certain distance metric to it". We showed that CRC constraints, ACCTC constraints, etc fit this characterization, and are therefore amenable to very simple polynomial-time randomized algorithms—the complexities of which were shown to be much less than that of the corresponding (known) deterministic algorithms and the (generic) lower bounds for establishing path-consistency. We also provided a simple polynomial-time algorithm for finding tree embeddings of variable domains (when they exist) for establishing tree convexity in path-consistent networks. Our future work is directed towards exploring other distance metrics that yield newer tractable classes of constraints, and even just simpler randomized algorithms for already established tractable classes of constraints. First thoughts on randomized algorithms for tree convex constraints appear in a small section in Chapter 2 of (Kumar 2005b). Applications of tree convex constraints (in the context of scene-labeling problems) can be found in (Zhang and Freuder 2004). Applications of CRC constraints in temporal reasoning problems can be found in (Kumar 2005c).

References

Booth K. S. and Lueker G. S. 1976. Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity using pq-Tree Algorithms. *J. Comput. Syst. Sci. 1976.*

Bulatov A. A., Krokhin A. A. and Jeavons P. G. 2000. Constraint Satisfaction Problems and Finite Algebras. *ICALP'00.*

Conitzer V., Derryberry J. and Sandholm T. 2004. Combinatorial Auctions with Structured Item Graphs. *AAAI'2004.*

Dechter R. 1992. Constraint Networks. *Encyclopedia of Artificial Intelligence, second edition, Wiley and Sons, pp 276-285, 1992.*

Dechter R. and Pearl J. 1991. Directed Constraint Networks: A Relational Framework for Causal Modeling. *IJCAI'91.*

Deville Y., Barette O. and Van Hentenryck P. 1999. Constraint Satisfaction over Connected Row-Convex Constraints. *Artificial Intelligence, 109:243-271.*

Doyle P. G. and Snell E. J. 1984. Random walks and Electrical Networks. *Carus Math. Monographs 22, Math. Assoc. Amer., Washington, D. C.*

Jeavons P. G., Cohen D. A. and Cooper M. 1998. Constraints, Consistency and Closure. *Artificial Intelligence, 101:251-265.*

Kumar T. K. S. 2005a. On the Tractability of Smooth Constraint Satisfaction Problems. *In Proceedings of CP-AI-OR'05.*

Kumar T. K. S. 2005b. Contributions to Algorithmic Techniques in Automated Reasoning about Physical Systems. *PhD Thesis, Computer Science Department, Stanford University, March 2005.*

Kumar T. K. S. 2005c. On the Tractability of Restricted Disjunctive Temporal Problems. *In Proceedings of ICAPS'2005.*

Mohr R. and Henderson T. C. 1986. Arc and Path Consistency Revisited. *Artificial Intelligence, 28:225-233.*

Papadimitriou C. H. 1991. On Selecting a Satisfying Truth Assignment. 1991. *In Proc. 32nd IEEE Symp. on the Foundations of Comp. Sci., pages 163-169, 1991.*

Van Beek P. and Dechter R. 1995. On the Minimality and Global Consistency of Row-Convex Constraint Networks. *JACM Archive Volume 42, Issue 3 (May 1995) Pages: 543 - 561.*

Van Hentenryck P., Deville Y. and Teng C. M. 1992. A Generic Arc-Consistency Algorithm and its Specializations. *Artificial Intelligence, 57:291-321.*

Zhang Y. and Freuder E. C. 2004. Tractable Tree Convex Constraint Networks. *AAAI'2004.*