Efficient Heuristic Approach to Dominance Testing in CP-nets

Minyi Li
Swinburne University of Technology
myli@swin.edu.au

Quoc Bao Vo
Swinburne University of Technology
BVO@swin.edu.au

Ryszard Kowalczyk
Swinburne University of Technology
RKowalczyk@swin.edu.au

ABSTRACT

CP-net (Conditional Preference Network) is one of the extensively studied languages for representing and reasoning with preferences. The fundamental operation of dominance testing in CP-nets, i.e. determining whether an outcome is preferred to another, is very important in many real-world applications. Current techniques for solving general dominance queries is to search for *improving flipping sequence* from one outcome to another as a proof of the dominance relation in all rankings satisfying the given CP-net. However, it is generally a hard problem even for binary-valued, acyclic CP-nets and tractable search algorithms exist only for specific problem classes. Hence, there is a need for efficient algorithms and techniques for dominance testing in more general problem settings. In this paper, we propose a heuristic approach, called DT*, to dominance testing in arbitrary acyclic multi-valued CP-nets. Our proposed approach guides the search process efficiently and allows significant reduction of search effort without impacting soundness or completeness of the search process. We present results of experiments that demonstrate the computational efficiency and feasibility of our approach to dominance testing.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems

General Terms

Algorithms, Design

Keywords

CP-nets; Dominance Testing; Heuristic

1. INTRODUCTION

The problems of eliciting, representing and reasoning with qualitative preferences over multi-attribute domain arise in many fields such as planning, design, and collective decision making [5, 6, 7, 8]. As the number of alternative outcomes of such domains is exponentially large in the number of attributes, it is unpractical to express preferences explicitly by giving out the ordering over the alternative outcome space. Therefore, the AI research community has developed languages for representing preferences in such domains in a succinct way, exploiting structural properties such as conditional preferential independence. The formalism of CP-nets (Conditional Preference Networks) [3] is among the most popular ones, since it preserves a good readability that is similar to the way users express their preferences in natural languages. CP-nets provide a compact representation of preference ordering in terms of natural preference statements under a *ceteris paribus* (all else being equal) interpretation. Ceteris paribus semantics induces a graph, known as *induced preference graph* [2, 3]; and an outcome α is said to dominate another outcome β if there exists a directed path, also called a *sequence of improving flips*, consisting of successively improving outcomes in the graph from β to α [1, 3]. Unfortunately, reasoning about the preference ordering (dominance relation) expressed by a CP-net is far from easy [3, 5]. With the exception of special cases such as CP-nets with tree or polytree structured conditional dependencies, dominance testing has been shown to be PSPACE-complete even with binary domain and acyclic dependences [5]. Some general pruning rules have been studied in [3] to reduce the search effort. But they might not be able to guide the search efficiently when the number of variables is large or the structure of the CP-net is complex. Another work proposed by Santhanam *et al.* [9] explores an approach to dominance testing with acyclic CP-nets via Model Checking. However, their approach mainly applies to binary-valued conditional preference statements. The complexity and feasibility of their approach to dominance testing in multi-valued CP-nets is still an open question. Hence, there is a need for efficient algorithms and techniques for dominance testing in more general problem settings.

To this end, we address the problem of dominance testing by proposing an efficient heuristic algorithm, called DT*, to guide the search process for improving flipping sequence from the worse outcome to the better outcome of the given query[1]. The proposed approach can be applied to arbitrary acyclic multi-valued CP-nets. It uses a numerical approximation of the given CP-net and considers the hamming distance between the currently considered outcome and the target outcome of the given query, i.e., the number of variables that the two outcomes differ from each other. We show that our proposed approach efficiently guides the search process for improving flipping sequence. It allows significant reduction of search effort without impacting *soundness* or *completeness* of the search process. Moreover, when there are no flipping sequences possible, it returns the quick failure for the dominance query without having to search all possible branches. We experimentally evaluate the proposed algorithm in different structure settings, including tree-structured CP-nets, directed-path singly connected CP-nets and arbitrary acyclic CP-nets, and with different domain sizes from binary to multi-valued. The experimental results presented in this

[1]The proposed heuristic will be described in the context of improving flipping sequences, but it can be applied to worsening search according to the same principle

Cite as: Efficient Heuristic Approach to Dominance Testing in CP-nets, Minyi Li, Quoc Bao Vo and Ryszard Kowalczyk, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. 353-360.

paper demonstrate that the proposed approach is computationally efficient. It allows dominance queries for CP-nets that are quite large and complex to be answered in reasonable time.

The remainder of this paper is organized as follows. Section 2 restates the necessary background on CP-nets and discusses some existing pruning techniques for the search process in dominance testing. Second 3 introduces the proposed approach in technical details and Section 4 presents the experimental results. Finally, Section 5 discusses the concluding remarks and outlines some directions for future research.

2. PRELIMINARIES

2.1 CP-nets

Let $\mathbf{V} = \{X_1, \ldots, X_n\}$ be a set of n variables, for each $X \in \mathbf{V}$, $D(X)$ is the *value domain* of X. A variable X is *binary* if $D(X) = \{x, \bar{x}\}$. If $\mathbf{X} = \{X_{i_1}, \ldots, X_{i_p}\} \subseteq \mathbf{V}$, with $i_1 < \cdots < i_p$ then $D(\mathbf{X})$ denotes $D(X_{i_1}) \times \cdots \times D(X_{i_p})$ and $\mathbf{x}$ denotes an assignment of variable values to $\mathbf{X}$ ($\mathbf{x} \in D(\mathbf{X})$). If $\mathbf{X} = \mathbf{V}$, $\mathbf{x}$ is a *complete assignment*; otherwise $\mathbf{x}$ is called a *partial assignment*. For any assignment $\mathbf{x} \in D(\mathbf{X})$, we denote by $\mathbf{x}[X]$ the value $x \in D(\mathbf{X})$ ($X \in \mathbf{X}$) assigned to variable X by that assignment; and $\mathbf{x}[\mathbf{W}]$ denotes the assignment of variable values $\mathbf{w} \in D(\mathbf{W})$ assigned to the set of variables $\mathbf{W} \subseteq \mathbf{X}$ by that assignment. If $\mathbf{x}$ and $\mathbf{y}$ are assignments to disjoint sets $\mathbf{X}$ and $\mathbf{Y}$, respectively ($\mathbf{X} \cap \mathbf{Y} = \emptyset$), we denote the combination of $\mathbf{x}$ and $\mathbf{y}$ by $\mathbf{xy}$. Let $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ be nonempty sets that partition $\mathbf{V}$ and $\succ$ a preference relation over $D(\mathbf{V})$, $\mathbf{X}$ is *(conditionally) preferentially independent* of $\mathbf{Y}$ given $\mathbf{Z}$ iff for all $\mathbf{x}, \mathbf{x}' \in D(\mathbf{X})$, $\mathbf{y}, \mathbf{y}' \in D(\mathbf{Y})$, $\mathbf{z} \in D(\mathbf{Z})$:

$$\mathbf{xyz} \succ \mathbf{x'yz} \text{ iff } \mathbf{xy'z} \succ \mathbf{x'y'z}$$

A CP-net $\mathcal{N}$ [3] over $\mathbf{V}$ is an annotated directed graph $\mathcal{G}$ over $X_1, \ldots, X_n$, in which nodes stand for the problem variables. Each node X is annotated with a conditional preference table (CPT), denoted by $CPT(X)$, which associates a total order $\succ^{X|\mathbf{u}}$ with each instantiation $\mathbf{u}$ of X's parents $Pa(X)$, i.e. $\mathbf{u} \in D(Pa(X))$. For instance, let $\mathbf{V} = \{X_1, X_2, X_3\}$, all three being binary, and assume that the preferences of an agent can be defined by a CP-net whose structural part is the directed acyclic graph $\mathcal{G} = \{(X_1, X_2), (X_1, X_3), (X_2, X_3)\}$; this means that the agent's preference over the values of X_1 is unconditional, preference over the values of X_2 (resp. X_3) is fully determined given the value of X_1 (resp. the values of X_1 and X_2). The preference statements contained in the conditional preference tables are written with the usual notation, that is, $x_1\bar{x}_2 : x_3 \succ \bar{x}_3$ means that when $X_1 = x_1$ and $X_2 = \bar{x}_2$ then $X_3 = x_3$ is preferred to $X_3 = \bar{x}_3$. Figure 1 illustrates an example of CP-net.

2.2 Dominance Testing

One of the most fundamental queries in any preference representation formalism is whether some outcome α dominates (i.e., is strictly preferred to) some other outcome β, called *Dominance Testing*. As discussed in [3, 9], such dominance queries in CP-nets are required whenever we wish to generate more than one non-dominated solutions to a set of hard constrains.

In this paper, we assume the structure of the CP-net is acyclic, i.e. does not contain any dependency cycles. In such case, two outcomes α and β can stand in one of three possible relations with respect to $\mathcal{N}$: either $\mathcal{N} \models \alpha \succ \beta$ (α is strictly preferred to β); or $\mathcal{N} \models \beta \succ \alpha$ (β is strictly preferred to α); or $\mathcal{N} \models \alpha \bowtie \beta$ (α and β are incomparable: $\mathcal{N} \not\models \alpha \succ \beta$ and $\mathcal{N} \not\models \beta \succ \alpha$). The third case means that the given CP-net $\mathcal{N}$ does not contain enough information to prove that either outcome is preferred to the other. Given an acyclic CP-net, comparisons between two outcomes that differ in the value of a single variable are easy: we only need to check the CPT of that variable and determine which outcome assigns it to a more preferred value. The better (improved) outcome can be considered as a product of a single *improving flip* in the value of a variable X from the worse outcome. For any pair of outcomes that differ on more than one variables, an outcome α is said to dominate another outcome β with respect to an acyclic CP-net $\mathcal{N}$ ($\mathcal{N} \models \alpha \succ \beta$) if there exists a *sequence of improving flips* from β to α. Otherwise, $\mathcal{N} \not\models \alpha \succ \beta$. The following definition of improving flipping sequence is introduced in [3].

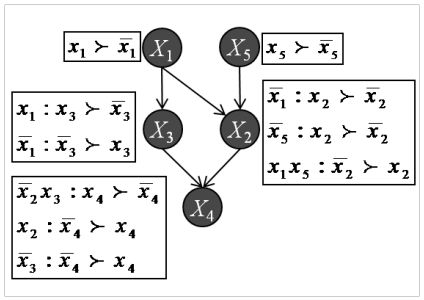

Figure 1: An example CP-net $\mathcal{N}$

DEFINITION 1 (IMPROVING FLIPPING SEQUENCE). *A sequence of outcomes $\beta = \gamma_1, \gamma_2, \ldots, \gamma_{m-1}, \gamma_m = \alpha$ such that*

$$\beta = \gamma_1 \prec \gamma_2 \prec \cdots \prec \gamma_{m-1} \prec \gamma_m = \alpha$$

is an improving flipping sequence *with respect to an acyclic CP-net $\mathcal{N}$ if and only if, $\forall 1 \leq i \leq m$, outcome γ_i is different from the outcome γ_{i+1} in the value of exactly one variable X, and $\gamma_{i+1}[X] \succ \gamma_i[X]$ given the parent context $\mathbf{u}$ of X assigned by γ_i and γ_{i+1}.*

For instance, consider the preference statements over two binary variables X_1 and X_2, $x_1 \succ \bar{x}_1$, $x_2 \succ \bar{x}_2$, the sequence $\bar{x}_1\bar{x}_2, \bar{x}_1x_2, x_1x_2$ is an improving flipping sequence from the outcome $\bar{x}_1\bar{x}_2$ to the best outcome x_1x_2.

2.3 Some General Search Techniques

Given an acyclic CP-net $\mathcal{N}$, a query $\mathcal{N} \models \alpha \succ \beta$ can be treated as a search for an improving flipping sequence from the less preferred outcome β to the more preferred outcome α. The search process can be implemented as an *improving search tree* rooted at β, $T(\beta)$. The children of every node[2] γ in $T(\beta)$ are those outcomes that can be reached by a single improving flip from γ. Consequently, every rooted path in $T(\beta)$ corresponds to some improving flipping sequence from the outcome β with respect to $\mathcal{N}$. Taking different directions in $T(\beta)$ leads to different improving sequences; however, taking a different direction during the tree traversal may also lead to a dead end, i.e., reach the optimal outcome of $\mathcal{N}$ without visiting the target outcome α of the query. Recent works have studied the computational complexity of testing dominance relations in CP-nets, e.g. [3, 5]. The results show that dominance testing in general CP-nets is PSPACE-complete and it remains PSPACE-complete even though the CP-net is acyclic [5]. Since the hardness of dominance testing, several search techniques for dominance queries have been studied in [3] in order to reduce the search effort.

Suffix Fixing. Let $X_{i_1} > \cdots > X_{i_n}$ be an arbitrary topological ordering consistent with the CP-net $\mathcal{N}$, an rth ($r \geq 1$)

[2]A node in the improving search tree is also an outcome.

suffix of an outcome α is the subset of the outcome values $\alpha[Xi_r]\,\alpha\left[X_{i_{r+1}}\right]\ldots\alpha[X_{i_n}]$. The rth suffix of outcomes α and β *match* iff $\forall r \leq j \leq n,\ \alpha\left[X_{i_j}\right] = \beta\left[X_{i_j}\right]$. For a query $\mathcal{N} \models \alpha \succ \beta$, suffix fixing rules out the exploration of any possible flipping sequences that destroy of the suffix of the currently considered outcome that matches the target outcome α. It prunes the subtree that improves the value of a variable within the matching suffix. For instance, consider the CP-net $\mathcal{N}$ in Figure 1 and the query $\mathcal{N} \models x_1\bar{x}_2x_3x_4x_5 \succ \bar{x}_1x_2x_3x_4\bar{x}_5$. Let $\alpha = x_1\bar{x}_2x_3x_4x_5$ and $\beta = \bar{x}_1x_2x_3x_4\bar{x}_5$, if pruned using suffix fixing and consider the variable ordering $X_1 > X_5 > X_2 > X_3 > X_4$, the 2^{nd} suffix x_3x_4 of α and β matches. Thus, the values of X_3 and X_4 will never be improved in the search tree $T(\beta)$, although given the assignment $\beta[X_1] = \bar{x}_1$ (resp. $\beta[X_2] = x_2$, $\beta[X_3] = x_3$), $\bar{x}_3 \succ x_3$ (resp. $\bar{x}_4 \succ x_4$). As shown in [3], any complete search algorithm for the improving search tree remains complete if pruning using suffix fixing is used.

Least-variable flipping. For every node γ in the improving search tree, least-variable flipping rule restricts flips to the variables that are least-improvable. Formally, a variable X is least-improvable in an outcome γ with respect to $\mathcal{N}$ if there is some value $x \in D(X)$ such that $x \succ_{\mathbf{u}} \gamma[X]$ (where $\mathbf{u} = \gamma[Pa(X)]$ is the parent context assigned by γ), and no descendent of X in γ has this property. For a query $\mathcal{N} \models \alpha \succ \beta$, *least-variable flipping rule* restricts attention to those variables that are not part of any matching suffix with the target outcome α and requires that the only neighbours of a node γ can be expanded in the search tree $T(\beta)$ are those in which some least improvable variable with respect to γ is improved.

However, least-variable flipping rule is only complete for a restricted class of CP-nets [3], i.e. tree-structured CP-nets and binary-valued, directed-path singly connected CP-nets. For multiply-connected networks, and networks with multi-valued variables, it does not guarantee completeness. That means, least-variable flipping may fail to find any improving sequence from β to α although there does exist at least one. In such case, it does not provide a correct answer to the given query. For instance, consider the CP-net $\mathcal{N}$ in Figure 1 and the query $\mathcal{N} \models x_1x_2x_3x_4x_5 \succ \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5$. [3] Starting with the root node $\beta = \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5$, the only least improvable variable that can be flipped is X_2. Unfortunately, flipping X_2 to value x_2 leads to outcome $\bar{x}_1x_2\bar{x}_3\bar{x}_4\bar{x}_5$, from which the target outcome $\alpha = x_1x_2x_3x_4x_5$ is unreachable. All branches in the improving search tree grow towards the optimal outcome $x_1\bar{x}_2x_3x_4x_5$ without going through the target outcome α of the query. Figure 2 shows the complete improving search tree $T(\beta)$ using least-variable flipping. However, there in fact exists a sequence of improving flips from β to α: $\bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5, x_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5, x_1\bar{x}_2x_3\bar{x}_4\bar{x}_5, x_1\bar{x}_2x_3x_4\bar{x}_5, x_1x_2x_3x_4\bar{x}_5, x_1x_2x_3x_4x_5$.

When the number of variables is large or the structure of the CP-net is complex, suffix fixing may not be able to guide the search efficiently while least-variable flipping rule does not guarantee completeness for general acyclic CP-nets. To this end, we will present another efficient heuristic approach to dominance testing. The proposed approach significantly prunes the search tree without impacting soundness or completeness of the search process.

[3]This example has also been discussed in Example 7 in [3]

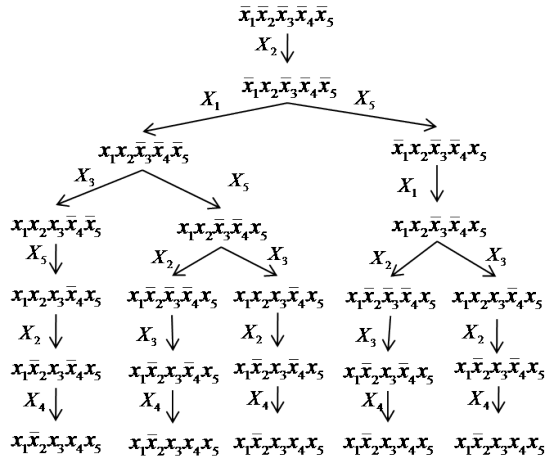

Figure 2: Improving search tree for query $\mathcal{N} \models x_1x_2x_3x_4x_5 \succ \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5$ using Least-variable flipping rule

3. HEURISTIC FOR DOMINANCE TESTING

In this section, we present our proposed heuristic approach, called DT*, to dominance testing in arbitrary acyclic CP-nets. In broad terms, we first define a penalty function based on a numerical approximation proposed by Domshlak *et al.* [4] that approximates acyclic CP-nets using weighted soft constraints. Then, an evaluation function is defined based on the hamming distance between the currently considered outcome and the target outcome and their penalties as a heuristic to guide the search process.

3.1 Penalty function

For a variable X, let $|D(X)|$ be the domain size of X and thus there are $|D(X)|$ degrees of penalties of X, denoted by $d_1, \ldots, d_{|D(X)|}$. Without loss of generality, we assume the degree of penalties of a variable X range between 0 and $|D(X)| - 1$; that is, $d_1 = 0, \ldots, d_{|D(X)|} = |D(X)| - 1$. For instance, consider the agent's CP-net in Figure 1, since all variables are binary, there are only two degrees of penalties, i.e., $d_1 = 0$ and $d_2 = 1$ for each variable. For a variable X, consider a preference ordering over the value of X given an instantiation of X's parents, let the rank of the most preferred value of X be 0 and the rank of the least preferred valued of X be $|D(X)| - 1$, given an outcome γ, the degree of penalty of a variable X in γ is then the rank of the value $\gamma[X]$ in the preference ordering over X given the parent context $\mathbf{u} = \gamma[Pa(X)]$. We denote by d_X^γ ($d_X^\gamma \in \left\{d_1, \ldots, d_{|D(X)|}\right\}$) the degree of penalty of X with respect to γ. For instance, consider a variable X such that $D(X) = \{x, x', x''\}$. Assume that, under a parent context $\mathbf{u} = \gamma[Pa(X)]$ assigned by an outcome γ, $x \succ x' \succ x''$. If $\gamma[X] = x$, then $d_X^\gamma = d_1 = 0$; if $\gamma[X] = x'$, then $d_X^\gamma = d_2 = 1$; if $\gamma[X] = x''$, then $d_X^\gamma = d_3 = 2$.

CP-net imposes a rich structure to allow variables to have different degrees of importance: variables "higher-up" in the structure of the network are considered to be more important than the lower level variables [1, 2, 3]. Thus, it is more important to obtain a preferred value for a variable than any of its descendents. We now analyse the *importance weight* of a variable in a CP-net. Given an acyclic CP-net $\mathcal{N}$ and consider an improving flip from an outcome γ to another outcome γ' that flips the value of a single variable X, changing the value of X may also affect the preference status of X's children. Thus, the resulting changes from γ to γ' includes: *(i)* the degree of penalty of X decreases from d_X^γ to $d_X^{\gamma'}$ ($d_X^\gamma > d_X^{\gamma'}$);

Algorithm 1: assgWeightCP ($\mathcal{N}$)

Input: $\mathcal{N}$, an acyclic CP-net
1 Order variables of $\mathcal{N}$ in a reverse topological ordering;
2 **foreach** $X \in \mathcal{N}$ **do**
3 **if** $Ch(X) = \emptyset$ **then**
4 $w_X \leftarrow 1$;
5 **else**
6 $w_X \leftarrow 1 + \sum_{Y \in Ch(X)} w_Y \cdot (|D(Y)| - 1)$;
7 **end**
8 **end**

and *(ii)* the degrees of penalty of X's children changes, which in the worst case, results in the degree of penalty of each chilren Y increasing from $d_Y^{\gamma} = d_1$ to $d_Y^{\gamma'} = d_{|D(Y)-1|}$. Consequently, in order to preserve the preference ordering induced by the given CP-net, the importance weight of a variable in that CP-net must be larger than the sum of the maximum penalties of its children. We now provide the formal definition of the variable importance weight in an acyclic CP-net.

DEFINITION 2 (IMPORTANCE WEIGHT). *Given an acyclic CP-net $\mathcal{N}$ over a set of variables $\mathbf{V}$. For each variable $X \in \mathbf{V}$, let $Ch(X)$ denote the set of children of X in $\mathcal{N}$, the importance weight of variable X, denoted by w_X, is recursively defined by:*

$$w_X = 1 + \sum_{Y \in Ch(X)} w_Y \cdot (|D(Y)| - 1) \quad (1)$$

Algorithm 1 provides a simple implementation to compute importance weights of variables. It takes *linear* time in the size of the network. Following a reverse topological ordering, it first assigns the importance weights to the variables that have no descendents (line 3–4) and then iteratively assigns the importance weights to the upper level variables according to Equation (1). Note that there are several ways to assign importance weights to the variables and the way we use here is different from [4]. In this paper, we consider the tight lower bound of the importance weight assignment, i.e. the sum of maximum penalties of the variable children $\sum_{Y \in Ch(X)} w_Y \cdot (|D(Y)| - 1)$.

EXAMPLE. Consider an agent's CP-net over a set of 5 variables $\mathbf{V} = \{X_1, \ldots, X_5\}$ in Figure 1. In this example, since all variables are binary, i.e. $\forall X \in \mathbf{V}$, $|D(X)| = 2$.We can assign the importance weight to each variable in a reverse topological ordering of variables: $w_{X_4} = 1$; $w_{X_2} = 1 + w_{X_4} \cdot (2-1) = 2$; $w_{X_3} = 1 + w_{X_4} \cdot (2-1) = 2$; $w_{X_1} = 1 + (w_{X_2} \cdot (2-1) + w_{X_3} \cdot (2-1)) = 5$; $w_{X_5} = 1 + w_{X_2} \cdot (2-1) = 3$. The importance weight of each variable in this CP-net is attached on top of the variables respectively in Figure 3.

Given an acyclic CP-net $\mathcal{N}$ and an outcome γ, the *penalty* of a variable X in γ is the *degree of penalty* of X in γ multiplied by the *importance weight* of X. The penalty of γ is then defined by the sum of penalties of the domain variables. We define the following penalty function for an acyclic CP-net based on the work by Domshlak *et al.* [4].

DEFINITION 3 (PENALTY FUNCTION). *Given an acyclic CP-net $\mathcal{N}$ over a set of variables $\mathbf{V}$ and an outcome γ. The penalty function pen, mapping from an outcome $\gamma \in O$ to $[0, +\infty]$, is defined as follows:*

$$\forall \gamma \in O,\ pen(\gamma) = \sum_{X \in \mathbf{V}} w_X \cdot d_X^{\gamma} \quad (2)$$

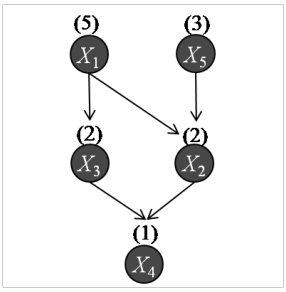

Figure 3: Variable importance weight in $\mathcal{N}$

EXAMPLE (CONT.) Consider our running example in Figure 1 and the outcome $\gamma = \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5$. As the agent unconditionally prefers $X_1 = x_1$ to $X_1 = \bar{x}_1$ (resp. $X_5 = x_5$ to $X_5 = \bar{x}_5$), $d_{X_1}^{\gamma} = 1$ (resp. $d_{X_5}^{\gamma} = 1$). On the other hand, $x_2 \succ \bar{x}_2$ (resp. $\bar{x}_3 \succ x_3$, $\bar{x}_4 \succ x_4$) given the parent context $X_1 = \bar{x}_1$ and $X_5 = \bar{x}_5$ (resp. $X_1 = \bar{x}_1$, $X_2 = \bar{x}_2$ and $X_3 = \bar{x}_3$) and thus $d_{X_2}^{\gamma} = 1$ (resp. $d_{X_3}^{\gamma} = 0$, $d_{X_4}^{\gamma} = 0$). Consequently, the penalty of outcome $\bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5$ is: $pen(\gamma) = w_{X_1} \cdot 1 + w_{X_2} \cdot 1 + w_{X_3} \cdot 0 + w_{X_4} \cdot 0 + w_{X_5} \cdot 1 = 5*1 + 2*1 + 2*0 + 1*0 + 3*1 = 10$.

In order to compute the penalty of an outcome, we simply need to sweep through the network from top to bottom (i.e., from ancestors to descendants), and to check the degree of penalty of the currently considered variable given its parent context. And finally we compute the penalty of the outcome based on Equation (2). Consequently, the penalty computation for a particular outcome takes *polynomial time* in the size of the network. We now prove that our algorithm for assigning penalties over alternative outcomes preserves the strict preference ordering induced by the original CP-net.

THEOREM 1. *Given an acyclic CP-net $\mathcal{N}$, we have:*

$$\forall \alpha, \beta \in O,\ \textit{if } \mathcal{N} \models \alpha \succ \beta \textit{ then } pen(\beta) > pen(\alpha)$$

PROOF. $\mathcal{N} \models \alpha \succ \beta$ if and only if there exists a sequence of improving flips from β to α, denoted by $Seq(\beta, \alpha) = \gamma_1 (= \beta), \gamma_2, \ldots, \gamma_{m-1}, \gamma_m (= \alpha)$, with respect to the conditional preference tables in $\mathcal{N}$. Each improving flip from γ_i to γ_{i+1} in $Seq(\beta, \alpha)$ that improves the value of a single variable X, $pen(\gamma_i) - pen(\gamma_{i+1}) = w_X \cdot \left(d_X^{\gamma_i} - d_X^{\gamma_{i+1}}\right) + \sigma$, where $\sigma \geq -\sum_{Y \in Ch(X)} w(Y) \cdot (|D(Y)| - 1)$ and $\left(d_X^{\gamma_i} - d_X^{\gamma_{i+1}}\right) \geq 1$. Thus, $pen(\gamma_i) - pen(\gamma_{i+1}) \geq w_X - \sum_{Y \in Ch(X)} w(Y) \cdot (|D(Y)| - 1) = w_X - (w_X - 1) = 1 > 0$. Consequently, with each improving flip from γ_i to γ_{i+1}, $pen(\gamma_i) > pen(\gamma_{i+1})$. Following from the transitivity: $pen(\gamma_1 (= \beta)) > pen(\gamma_2) > \cdots > pen(\gamma_{m-1}) > pen(\gamma_m) (= \alpha)$ and thus $pen(\beta) > pen(\alpha)$. □

COROLLARY 1. *Given an acyclic CP-net $\mathcal{N}$, $\forall \alpha, \beta \in O$,*

- *if $pen(\beta) > pen(\alpha)$ then $\mathcal{N} \models \alpha \succ \beta$ or $\mathcal{N} \models \alpha \bowtie \beta$*
- *if $pen(\beta) = pen(\alpha)$ then $\mathcal{N} \models \alpha \bowtie \beta$*

LEMMA 1. *Given an acyclic CP-net $\mathcal{N}$ over a set of variables $\mathbf{V}$,let α, β be any pair of outcomes that $\mathcal{N} \models \alpha \succ \beta$; $\mathbf{IS}$ the set of all possible improving flipping sequence from β to α with respect to the CPTs in $\mathcal{N}$; $HD(\beta, \alpha)$ the hamming distance between β*

*and α (Note that both in binary-valued and multi-valued CP-nets, the hamming distance is defined by the number of variables that the two outcomes differ from each other.); $Seq(\beta, \alpha) \in$ **IS** an improving flipping sequence from β to α; $|Seq(\beta, \alpha)|$ is the length of $Seq(\beta, \alpha)$ and thus the number of improving flips from β in this sequence is $|Seq(\beta, \alpha)| - 1$, then,*

$$HD(\beta, \alpha) \leq |Seq(\beta, \alpha)| - 1 \leq pen(\beta) - pen(\alpha)$$

PROOF. As each improving flip flips the value of a single variable, if $\mathcal{N} \models \alpha \succ \beta$, there must be at least $HD(\beta, \alpha)$ flips that flips the value of each variable X that β and α differ, from $\beta[X]$ to $\alpha[X]$. Thus, $\forall Seq(\beta, \alpha) \in$ **IS**, $|Seq(\beta, \alpha)| - 1 \geq HD(\beta, \alpha)$. On the other hand, any improving flip from γ_i to γ_{i+1} in $Seq(\beta, \alpha)$ that flips the value of a single variable X, $pen(\gamma_i) - pen(\gamma_{i+1}) \geq 1$ (see the proof of Theorem 1). Thus, $pen(\gamma_i) - pen(\gamma_{i+1}) \geq 1$. Assume γ is an outcome in $Seq(\beta, \alpha)$ that improved from β by t flips, $pen(\beta) - pen(\gamma) \geq t$ and $pen(\beta) - t \geq pen(\gamma)$. Thus, $pen(\beta) - pen(\alpha) - t \geq pen(\gamma) - pen(\alpha)$. If $t > pen(\beta) - pen(\alpha)$, then $pen(\beta) - pen(\alpha) - t < 0$ and $pen(\gamma) - pen(\alpha) < 0$. According to Corollary 1, $\mathcal{N} \models \gamma \succ \alpha$ or $\mathcal{N} \models \gamma \bowtie \alpha$ ($\mathcal{N} \not\models \alpha \succ \gamma$), contradicting the fact that γ is in the improving sequence $Seq(\beta, \alpha)$. Hence, the number of improving flips from β in $Seq(\beta, \alpha)$ can not be greater than $pen(\beta) - pen(\alpha)$, $|Seq(\alpha, \beta)| - 1 \leq pen(\beta) - pen(\alpha)$. □

3.2 The proposed DT* algorithm

The penalty function mentioned above provides an order-preserving numerical approximation for a given CP-net. We also show the upper bound and lower bound of the number of improving flips from a worse outcome to a preferred outcome in Lemma 1. In this section, these results are used as a heuristic in the search process for improving flipping sequence. The proposed algorithm has a number of desirable properties:

- it often returns the quick failure for the dominance query if no flipping sequence is possible;
- it often quickly shows that back-tracking is needed when there is no possible flipping sequence to the target outcome following the currently considered path; and,
- it efficiently guides the search direction without compromising soundness or completeness of the search process.

Given an acyclic CP-net $\mathcal{N}$ and a pair of outcomes α and β, for the query $\mathcal{N} \models \alpha \succ \beta$, we build the search tree $T(\beta)$ and search for an improving flipping sequence to the target outcome α as discussed in [3]. We introduce the evaluation function f for the heuristic search strategy as follows:

DEFINITION 4 (EVALUATION FUNCTION). *Given an acyclic CP-net $\mathcal{N}$ and the query $\mathcal{N} \models \alpha \succ \beta$ ($\alpha, \beta \in O$). The evaluation function f, mapping from a node (i.e., an outcome) γ in the improving search tree $T(\beta)$ to $[0, +\infty]$, is defined by:*

$$f(\gamma) = pen(\gamma) - HD(\gamma, \alpha) - pen(\alpha) \quad (3)$$

Our proposed heuristic algorithm DT* (see Algorithm 2) is adapted from the A* heuristic search algorithm with $f(\gamma)$ being the evaluation function. It maintains a priority queue of nodes to be expanded, known as the $fringe$. On the one hand, the lower f value for a node γ, the higher its priority is. On the other hand, we only consider the outcomes that the f value is non-negative. That means,

Algorithm 2: DT*($\mathcal{N} \models \alpha \succ \beta$)

Input: a dominance query (an acyclic CP-net $\mathcal{N}$; a pair of outcomes α and β; and determining whether $\mathcal{N} \models \alpha \succ \beta$)
Output: $True$: $\mathcal{N} \models \alpha \succ \beta$; $False$: $\mathcal{N} \not\models \alpha \succ \beta$

```
1  if f(β) < 0 then
2      return False;
3  else
4      fringe ← INSERT(MAKE-NODE(β), fringe);
5      while fringe ≠ ∅ do
6          γ* ← REMOVE-FIRST(fringe);
7          if GOAL-TEST(γ* = α) then
8              return True;
9          else
10             foreach X ∈ N do
11                 if IMPROVABLE(γ*, X)
                   &&X ∉ ANY-MATCHING-SUFFIX(γ*, α)
                   then
12                     γ' ← SINGLE-FLIP(γ*, X);
13                     if NOT-REPEATED(γ') && f(γ') ≥ 0
                       then
14                         INSERT-ASC(MAKE-NODE(γ'), fringe)
15                     end
16                 end
17             end
18         end
19     end
20     return False
21 end
```

an outcome γ will be added into the $fringe$ only if $f(\gamma) > 0$. In essence, an outcome with a negative f value means that there is no possible improving flipping sequence from that outcome to the target outcome α (see Lemma 2). Before adding the original node β into the $fringe$, the f value of β will be computed and the algorithm will return $False$ if $f(\beta) < 0$ (line 1–2). In this case, the query fails ($\mathcal{N} \not\models \alpha \succ \beta$) even before building the root node of the improving search tree. Otherwise, β will be added into the $fringe$ as the root node of the improving search tree $T(\beta)$ (line 4). At each iteration of DT*, the first node γ^*, i.e. the node with the lowest f value, is removed from the $fringe$ and being expanded (line 4). The children of a node in $T(\beta)$ are those outcomes that can be reached by a single improving flip from that node. Our proposed algorithm applies suffix fixing rules, restricting attention to those variables in γ^* that are not part of any matching suffix with the target outcome α (line 11). Moreover, it requires that a child γ' of a node be added into the $fringe$ if and only if: *(i)* γ' has not been traversed before; and *(ii)* $f(\gamma') \geq 0$ (line 13). For the current node γ^* under consideration, we add each child γ' of γ^* that meets the above requirements into the $fringe$ in ascendant order of the f values of the nodes in the $fringe$ (line 14). DT* continues until: the currently considered node for expansion equals to the target outcome α, then it ends and returns $True$ ($\mathcal{N} \models \alpha \succ \beta$) (line 7–8); or the $fringe$ is empty, it returns $False$ ($\mathcal{N} \not\models \alpha \succ \beta$) (line 20).

In order to prove the completeness of our proposed heuristic algorithm, we first proof the follow lemma.

LEMMA 2. *Given an acyclic CP-net $\mathcal{N}$ and a query $\mathcal{N} \models \alpha \succ \beta$ ($\alpha, \beta \in O$), $\forall \gamma^* \in O$, if $f(\gamma^*) < 0$, then γ^* would not be part of any possible improving flipping sequence from β to α.*

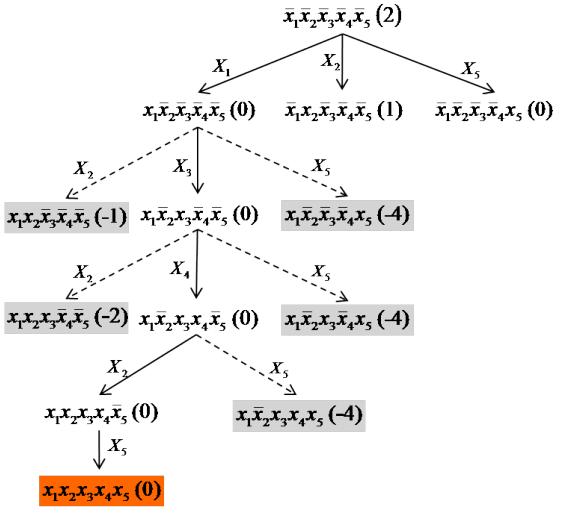

Figure 4: Improving search tree

PROOF. During the execution of DT* algorithm, for any outcome γ^* (including β), $f(\gamma^*) = pen(\gamma^*) - HD(\gamma^*, \alpha) - pen(\alpha)$. Assume that there exist an improving flipping sequence $Seq(\gamma^*, \alpha) = \gamma_1(=\gamma^*), \gamma_2, \ldots, \gamma_{m-1}, \gamma_m(=\alpha)$ from γ^* to the target outcome α. Based on Lemma 1, we know that there must be at least $HD(\gamma^*, \alpha)$ flips improved from γ^*. For any improving flip from γ_i to γ_{i+1}, $pen(\gamma_i) - pen(\gamma_{i+1}) \geq 1$. Consequently, for any outcome γ' that improved from γ^* by $HD(\gamma^*, \alpha)$ flips, $pen(\gamma^*) - pen(\gamma') \geq HD(\gamma^*, \alpha)$ and thus $pen(\gamma^*) - HD(\gamma^*, \alpha) \geq pen(\gamma')$. Hence, $pen(\gamma^*) - HD(\gamma^*, \alpha) - pen(\alpha) \geq pen(\gamma') - pen(\alpha)$. Because $f(\gamma^*) < 0$, $pen(\gamma') - pen(\alpha) \leq pen(\gamma^*) - HD(\gamma^*, \alpha) - pen(\alpha) < 0$. Consequently, $pen(\gamma') - pen(\alpha) < 0$ and $\mathcal{N} \not\models \alpha \succ \gamma'$. γ' will not be part of any possible improving flipping sequence to α, contradicting the fact that there exist an improving flipping sequence $Seq(\gamma^*, \alpha)$ from γ^* to the target outcome α. □

We now prove the completeness of our proposed heuristic algorithm.

THEOREM 2. *DT* is complete for any arbitrary acyclic CP-nets.*

PROOF. DT* traverses the tree searching all neighbours; it follows lowest evaluated value path and keeps a sorted priority queue of alternate path segments along the way. If at any point the path being followed has a higher evaluated value than other encountered path segments, the higher evaluated value path is kept in the $fringe$ and the process is continued at the lower value sub-path. This continues until the currently considered node for expansion is the target outcome or the fringe is empty. During the execution of DT* algorithm, there are three kinds of nodes will be pruned: *(i)* the outcomes that have been explored previously; *(ii)* the outcomes that improve the value of the variable that is part of some matching suffix with the target outcome; and *(iii)* the outcomes with negative f values. Obviously, checking repeated nodes does not affect the completeness of the algorithm. Also, as shown in [3], any complete search algorithm for the improving search tree remains complete if pruning using suffix fixing rule is used. Furthermore, we have already proved in Lemma 2 that an outcome γ^* with $f(\gamma^*) < 0$ will not be part of any possible improving sequence from β to α, so pruning the third kind of outcomes also does not affect the completeness of the algorithm. Consequently, DT* is complete for any acyclic CP-nets. □

EXAMPLE (CONT.) We now demonstrate the execution of DT* algorithm with the CP-net in our running example (Figure 1) and consider the query $\mathcal{N} \models x_1x_2x_3x_4x_5 \succ \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5$. Let $\alpha = x_1x_2x_3x_4x_5$ and $\beta = \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5$, we first consider the f value of the less preferred outcome β of the query. As $f(\beta) = pen(\beta) - HD(\beta, \alpha) - pen(\alpha) = 10 - 5 - 3 = 2 > 0$, we build the search tree $T(\beta)$ with β being the root node and add β into the $fringe$.
In the 1^{th} iteration of DT*, $\gamma^* = \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5$ is removed from the $fringe$ to be expanded. There are three improvable variable from γ^*: X_1, X_2 and X_5. Hence, there are three children nodes: $x_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5$, $\bar{x}_1x_2\bar{x}_3\bar{x}_4\bar{x}_5$ and $\bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4x_5$. The f value of these three children nodes are computed accordingly. $f(x_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5) = 0$, $f(\bar{x}_1x_2\bar{x}_3\bar{x}_4\bar{x}_5) = 1$ and $f(\bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4x_5) = 0$. As none of the f value of these three children nodes is negative, all of them are added into the $fringe$ according to the ascendant order of the f value.
In the 2^{nd} iteration, the first outcome $\gamma^* = x_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5$ with the lowest f value is removed from the $fringe$ (Assume that the nodes with the same f value will be traversed in the order from left to right). There are three possible children nodes of γ^*: $x_1x_2\bar{x}_3\bar{x}_4\bar{x}_5$, $x_1\bar{x}_2x_3\bar{x}_4\bar{x}_5$ and $x_1\bar{x}_2\bar{x}_3\bar{x}_4x_5$. As $f(x_1x_2\bar{x}_3\bar{x}_4\bar{x}_5) = 5-3-3 = -1 < 0$; $f(x_1\bar{x}_2x_3\bar{x}_4\bar{x}_5) = 6-3-3 = 0$; and $f(x_1\bar{x}_2\bar{x}_3\bar{x}_4x_5) = 2-3-3 = -4 < 0$. There is only one outcome $x_1\bar{x}_2x_3\bar{x}_4\bar{x}_5$ will be added into the $fringe$.
In the 3^{rd} iteration, we continue with the outcome $\gamma^* = x_1\bar{x}_2x_3\bar{x}_4\bar{x}_5$. There are three possible outcomes can be reached by a single flip from γ^*: $x_1x_2x_3\bar{x}_4\bar{x}_5$, $x_1\bar{x}_2x_3x_4\bar{x}_5$ and $x_1\bar{x}_2x_3\bar{x}_4x_5$. We compute the f value of these three outcomes: $f(x_1x_2x_3\bar{x}_4\bar{x}_5) = 3 - 2 - 3 = -2 < 0$; $f(x_1\bar{x}_2x_3x_4\bar{x}_5) = 5 - 2 - 3 = 0$; and $f(x_1\bar{x}_2x_3\bar{x}_4x_5) = 1 - 2 - 3 = -4 < 0$. Only one outcome $x_1\bar{x}_2x_3x_4\bar{x}_5$ can be added into the $fringe$.
Similarly, in the 4^{th} iteration, we explore the outcome $\gamma^* = x_1\bar{x}_2x_3x_4\bar{x}_5$ and add only one outcome $x_1x_2x_3x_4\bar{x}_5$ into the $fringe$.
In the 5^{th} iteration, we explore the outcome $\gamma^* = x_1x_2x_3x_4\bar{x}_5$. In essence, there are two variables can be improved from γ^*: X_4 and X_5. However, as X_4 is in the 3^{rd} matching suffix with the target outcome α (using the topological order $X_1 > X_5 > X_2 > X_3 > X_4$), we only consider flipping the value of X_5. And this step produces the target outcome α, which will be explored in the last iteration and the algorithm returns $True$ to this query.
Note that as we have discussed in Section 2.3, an algorithm based on Least-variable flipping rule is incomplete in this case.

4. EXPERIMENT

We now describe the results of experiments that show the feasibility of our approach to dominance testing with respect to *(i)* the average number of visited nodes during the search process; *(ii)* the number of variables s_{var} and the domain size s_{dz} that can be efficiently handled in practice; and *(iii)* the structure of CP-nets. We compare the performance of the proposed DT* algorithm with *(i)* a standard depth-first search algorithm that applies suffix fixing during the search, called DF; and *(ii)* an algorithm using Least-variable flipping rule, called LVF. We generate random preference networks by varying the number of variables, the structure of the network and the preference of the variables. For directed-path singly connected CP-nets and arbitrary acyclic CP-nets, we restrict the maximum in-degree of each node in the generated CP-nets to 10. For multi-valued CP-nets, we restrict the maximum domain size s_{dz} to 5. We conduct the following six sets of experiments. At each set of experiments, we generate 1000 CP-nets randomly and using each resulting preference network, we evaluate 5 dominance queries by picking distinct pairs of outcomes at random.

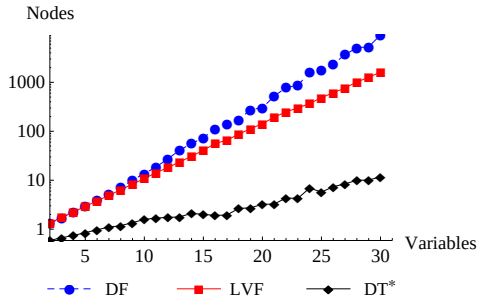

Figure 5: Avg. number of visited nodes with binary-value tree-structured CP-nets

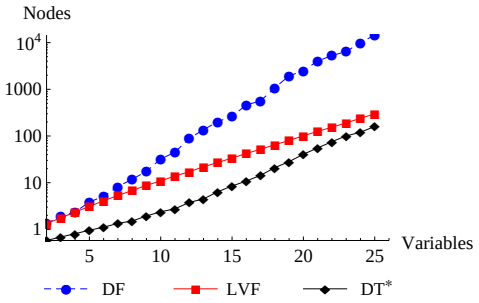

Figure 7: Avg. number of visited nodes with binary-valued, directed-path singly connected CP-nets

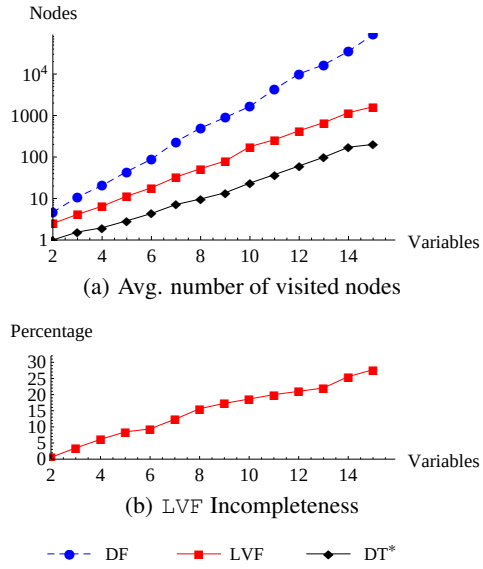

Figure 6: Multi-valued tree-structured CP-nets

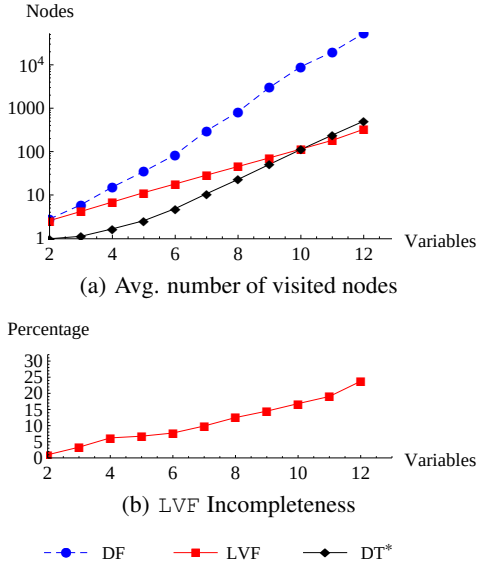

Figure 8: Multi-valued polytree CP-nets

Set 1: binary-valued tree-structured CP-nets. We vary the number of variables s_{var} from 2 to 30 and only generate tree-structured dependences. From Figure 5 we can observe that on average, the numbers of visited nodes by both DT* and LVF algorithms are much less than DF algorithm. Note that for binary-valued tree-structured CP-nets, LVF (Least-variable flipping rule) is guaranteed to be *complete* and *backtrack-free*. [4] However, on average, DT* is more efficient than the LVF algorithm for dominance testing in tree-structured CP-nets. The average execution time of DT* approach with 30 variables is less than 0.03 seconds. It offers more than three orders of magnitude improvement in performance over the DF algorithm.

Set 2: multi-valued tree-structured CP-nets. We vary s_{var} from 2 to 15. The results of multi-valued tree-structured CP-nets (see Figure 6(a)) is similar to the set of experiments with binary-valued tree-structured CP-nets. However, LVF algorithm does not guarantee completeness in multi-valued CP-nets. Figure 6(b) shows the percentage of cases in which the LVF algorithm is incomplete, i.e., it gives an incorrect answer to the query. In general, the percentage of incompleteness of LVF algorithm is increasing as the number of variables increases. When there are 15 variables, in more than 28% cases that LFV algorithm fails to find the improving flipping sequence for the given query although there does exist at least one. On the other hand, according to the experiment data, DT* completes the search process in about 12 seconds on average in the cases of 15 variables.

Set 3: binary-valued, directed-path singly connected CP-nets. In this set of experiments, the number of variables s_{var} is from 2 to 25. Note that LVF algorithm guarantees completeness in binary-valued, directed-path singly connected CP-nets while it may require back-tracking during the search. The average number of visited nodes in this set of experiments is shown in Figure 7. Both LVF and DT* algorithms are much more efficient than the DF algorithm. When there are 25 variables, the average execution time of DT* is about 5.7 seconds, which is more than two orders of magnitude less than the DF algorithm.

Set 4: multi-valued, directed-path singly connected CP-nets. We vary s_{var} from 2 to 12. Figure 8(a) shows that the average number of visited nodes of both LVF and DT* algorithms are much less than DF algorithm. Although the result shows that

[4]The authors can also refer to [3] Page 161, TreeDT algorithm for binary-valued, tree-structured CP-nets

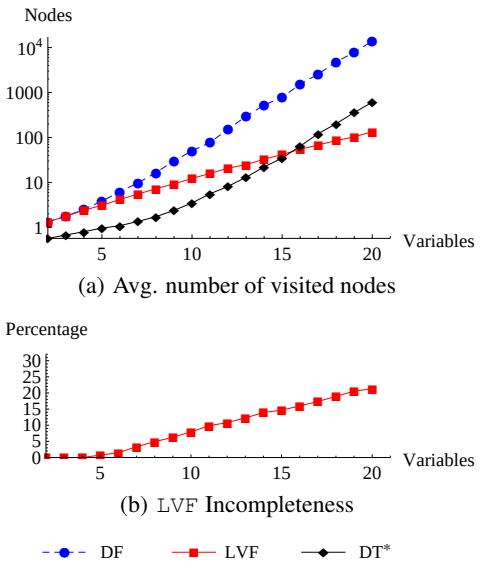

Figure 9: Binary-value arbitrary acyclic CP-nets

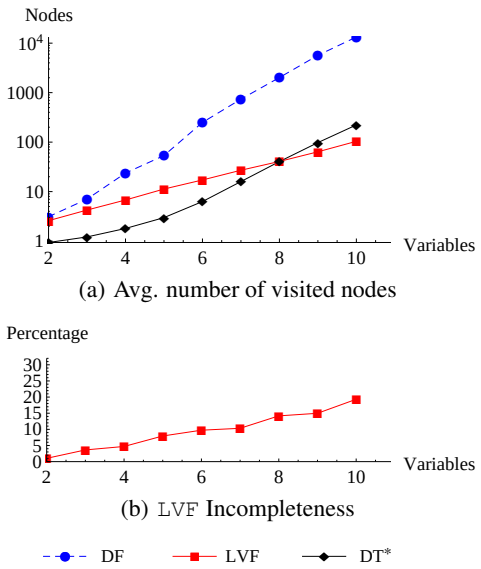

Figure 10: Multi-valued arbitrary acyclic CP-nets

when the number of variables is large, the LVF algorithm may visit less nodes than DT* algorithm, the percentage of incompleteness of LVF is on the other hand, increasing as the number of variables increases (see Figure 8(b)). When there are 12 variables, this percentage is more than 25%. According to the experimental data, with 12 variables and each variable with the maximum domain size of 5, the average execution time of DT* approach is still less than 50 seconds.

Set 5: binary-valued arbitrary acyclic CP-nets. We vary s_{var} from 2 to 20. Similar to the results presented in Set 4, when the number of variables is large (more than 15) the average number of visited nodes of DT* algorithm is more than that of LVF algorithm (see Figure 9(a)). However, for binary-valued CP-nets in general, LVF does not guarantee completeness and the percentage of cases that the LVF algorithm returns incorrect answers is increasing as the number of variable increases (Figure 9(b)). When there are 20 variables, this percentage is more than 20%. While on average, DT* algorithm returns a correct answer to the given query in about 20 seconds.

Set 6: multi-valued arbitrary acyclic CP-nets. In the last set of experiments, we vary s_{var} from 2 to 10. The results with arbitrary acyclic CP-nets in multi-valued setting is similar to that in binary-valued setting (see Figure 10(a) and Figure 10(b)). When there are 10 variables, the percentage of incomplete cases the LVF algorithm is more than 20%; on the other hand, DT* guarantees to return a correct answer in about 9 seconds on average.

In summary, our experiments show that on average, our proposed DT* algorithm is much more efficient than the DF algorithm. It is as relatively efficient as LVF algorithm while guaranteeing soundness and completeness of the search process. From the experiment, we can also conclude that our proposed DT* algorithm allows dominance queries for CP-nets that are quite large and complex to be answered in reasonable time.

5. CONCLUSION AND FUTURE WORK

In this paper, we have studied the problem of dominance testing in CP-nets. We have proposed a heuristic algorithm DT* for dominance testing with arbitrary acyclic CP-nets. The proposed approach significantly reduces the search effort without impacting soundness and completeness. We have also experimentally shown that the proposed algorithm is computationally efficient.

Nonetheless, the present work is only applicable for acyclic CP-nets. The investigation of techniques to deal with cyclic preferences need to be further explored.

6. ACKNOWLEDGEMENTS

This work is partially supported by the ARC Discovery Grant DP0987380.

7. REFERENCES

[1] C. Boutilier, R. I. Brafman, H. H. Hoos, and D. Poole. Reasoning with conditional ceteris paribus preference statements. In *UAI*, pages 71–80. Morgan Kaufmann Publishers, 1999.

[2] C. Boutilier, R. I. Brafman, H. H. Hoos, and D. Poole. Preference-based constrained optimization with CP-nets. *Computational Intelligence*, 20:137–157, 2001.

[3] C. Boutilier, R. I. Brafman, H. H. Hoos, and D. Poole. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21:135–191, 2004.

[4] C. Domshlak, S. D. Prestwich, F. Rossi, K. B. Venable, and T. Walsh. Hard and soft constraints for reasoning about qualitative conditional preferences. *J. Heuristics*, 12(4-5):263–285, 2006.

[5] J. Goldsmith, J. Lang, M. Truszczynski, and N. Wilson. The computational complexity of dominance and consistency in CP-nets. *J. Artif. Int. Res.*, 33(1):403–432, 2008.

[6] C. Lafage and J. Lang. Logical representation of preferences for group decision making. In *KR*, pages 457–468, 2000.

[7] J. Lang. Graphical representation of ordinal preferences: Languages and applications. In *ICCS*, pages 3–9, 2010.

[8] F. Rossi, K. B. Venable, and T. Walsh. mCP nets: Representing and reasoning with preferences of multiple agents. In *AAAI*.

[9] G. R. Santhanam, S. Basu, and V. Honavar. Dominance testing via model checking. In *AAAI*, 2010.