Extended Semantics and Optimization Algorithms for CP-Networks

Ronen I. Brafman
Dept. of Computer Science
Ben-Gurion University
Beer-Sheva 84105, Israel
brafman@cs.bgu.ac.il

Yannis Dimopoulos
Dept. of Computer Science
University of Cyprus
Nicosia 1678, Cyprus
yannis@cs.ucy.ac.cy

December 18, 2003

Abstract

Preference elicitation is a serious bottleneck in many decision support applications and agent specification tasks. CP-nets were designed to make the process of preference elicitation simpler and more intuitive for lay users by graphically structuring a set of *Ceteris Paribus* (CP) preference statements – preference statements that most people find natural and intuitive. Beside their usefulness in the process of preference elicitation, CP-nets support efficient optimization algorithms which are crucial in most applications (e.g., the selection of the best action to execute or the best product configuration). In various contexts, CP-nets with an underlying cyclic structure emerge naturally. Often, they are inconsistent according to the current semantics, and the user is required to revise them. In this paper we show how optimization queries can be meaningfully answered in many "inconsistent" networks without troubling the user with requests for revisions. In addition, we describe a method for focusing the user's revision process when revisions are *truly* needed. In the process, we provide a formal semantics that justifies our approach and new techniques for computing optimal outcomes. Some of the methods we use are based on a reduction to the problem of computing stable models for nonmonotonic logic programs, and we explore this relationship closely.

1 Introduction

Ceteris Paribus (CP) preference statements are among the most natural and most intuitive preference statements that people make. Thus, it is not surprising that they have drawn the attention of many researchers in philosophy and AI (e.g., [12, 17]). CP statements indicate a preference for one value over another in the context of a fixed background. For example, the statement "I prefer an apple pie to a chocolate cake as a dessert, *ceteris paribus*" expresses the fact that given two identical contexts – i.e., meals – that differ only in their dessert, the one containing an apple pie is preferred to the one containing a chocolate cake. Finer distinctions can be made using conditional CP statements. For example: "I prefer red wine over white wine if the main course is beef." In this case, the preference for red wine to white wine is restricted to comparisons between identical meals in which the main course is beef.

CP-nets [3] are a graphical tool for representing and for structuring a set of CP statements. They were motivated by the need to provide a tool that would support a simple preference elicitation process that does not require the presence of a decision analyst and could be used for various online applications, among other things. A CP-network consists of a graph describing the preferential dependency relationship between different domain variables. Each node is annotated by a conditional preference table (CPT) that describes how the user's preference over the different values of the variable associated with this node depends on the variables associated with the parents of this node. Although CP-nets are quite a recent development, they play a key role in a number of recent applications (e.g., see [11, 16, 4].

Cyclic CP-nets emerge naturally when there is a set of interdependent variables, none of which is more important than the other. For example, [11] note that such dependency can emerge among web-page components in their web-personalization tool. Cyclic CP-network raise some conceptual and computational problems to which we still do not have satisfactory answers. Even worse, according to the standard semantics of CP-nets, most cyclic CP-nets are inconsistent. For example, in [10], it was shown that the preference ordering induced by any simple cycle (i.e., a cycle that does not contain smaller cycles) with more than two nodes is inconsistent. That is, there will be at least two outcomes, o_1 and o_2, such that one can show that o_1 is strictly preferred to o_2 and o_2 is strictly preferred to o_1.

The fact that many cyclic networks are inconsistent raises a serious practical concern. When a user specifies an inconsistent cyclic network, we must ask him to revise his network. To do so, we need to provide information that will help him obtain a consistent network. In this paper we make two practical contributions aimed at improving this process. First, we show that various optimization queries can be answered naturally even when the network is "inconsistent" according to the standard semantics. In such cases, no additional burden is placed on the user. Second, when revision is required, we show how the notion of a partial model, studied in the area of logic programming, can be used to identify those aspects of a model that require revision.

While pursuing our more practical goals we make a number of technical and semantic contributions. First, we provide a more flexible semantics for CP-nets that is identical with the current semantics on those networks the latter considers consistent. This semantics justifies our approach for generating optimal outcomes given networks that are inconsistent under the standard semantics. Second, we show how to answer optimization queries using various approaches that include reduction to SAT, the use of cycle cut-sets, and reduction to logic programs. This extends current method which are restricted to acyclic nets, and helps us define the notion of a partial outcome. It also provides an interesting link between work on CP-nets and other nonmonotonic formalisms. We take a particularly close look at the relationship between the *stable models* semantics for nonmonotonic logic programs, also referred to as the *answer set* semantics, and CP-networks. Finally, we compare CP-networks and the closely related formalism of *Answer Set Optimization* [6].

The paper is structured as follows: In Section 2 we provide the necessary background on ceteris paribus preference statements and CP-nets. In Sections 3 we take a closer look at the notion of consistency in CP-nets and the problem that cyclic networks introduce. We suggest an extension of the current semantics and a natural definition of an optimal model as one that cannot be improved. In Section 4 we discuss a number of techniques for computing optimal models according to our semantics. In Section 5 we discuss an alternative translation into nonmonotonic logic programs and some of its implications. In Section 6 we continue to explore the relationship between the stable model semantics and CP-nets and examine the problem of credulous and skeptical reasoning with CP-nets. In Section 7 we discuss related work, in particular, *Answer-Set Optimization*. We conclude in Section 8.

2 CP-Nets

We start with a review of *ceteris paribus* preference statements and preferential independence, followed by the definition of CP-nets.

2.1 Ceteris Paribus Preference Statements

CP-nets are a tool for specifying preference relations. Formally, a preference relation over a set S is a binary relation over S, i.e., a subset of $S \times S$. We can interpret the fact that (o, o') belongs to this subset in two ways. The first is that o is *at least as preferred as* o' (a.k.a. o is *weakly preferred* to o'). This is denoted $o \succeq o'$. Alternatively, we can interpret this as indicating that o is *strictly preferred* to o', and this is denoted by $o \succ o'$. This distinction plays an important role in the semantics proposed in this paper.

Preference relations are binary relations with some additional properties: the relation $\succeq$ is a partial pre-order, and the relation $\succ$ is a partial order. A *partial pre-order* is a reflexive and transitive binary relation. That is, $o \succeq o$ holds, and $o \succeq o'$, $o' \succeq o''$ imply $o \succeq o''$. A (strict) partial order is a transitive and anti-symmetric binary relation. The relation $\succ$ is anti-symmetric if $o \succ o'$ implies $o' \not\succ o$. Notice that anti-symmetry implies, in particular, irreflexivity.

Given a partial pre-order $\succeq$, we can define the induced strict partial order as follows: $o \succ o'$ iff $o \succeq o'$ and $o' \not\succeq o$. It is easy to see that this relation is, indeed, transitive and anti-symmetric.

All the above definitions extend to the case of a *total* pre-order by requiring that for every $o, o' \in S$, at least one of $o \succeq o'$ or $o' \succeq o$ holds. A total (strict) order is a (strict) order in which for every distinct $o, o' \in S$ exactly one of $o \succ o'$ or $o' \succ o$ must hold.

A total pre-order is often referred to as a *ranking* because it induces a natural total order over equivalence classes of elements of S. $o, o' \in S$ belong to the same equivalence class iff $o \succeq o'$ and $o' \succeq o$. The rank of o is then the level of its equivalence class. Notice, though, that the fact that $\succeq$ is a total pre-order does not imply that the strict relation $\succ$ that it induces is a total order: If o and o' are equally preferred according to $\succeq$, i.e., $o \succeq o'$ and $o' \succeq o$, then neither $o \succ o'$ nor $o' \succ o$ holds. Thus, the strict binary relation induced by a total pre-order may be partial, but must be transitive and anti-symmetric.

The types of outcomes we are concerned with consist of possible assignments to some set of variables. More formally, we assume some given set $\mathbf{V} = \{X_1, \ldots, X_n\}$ of variables with corresponding domains $\mathcal{D}(X_1), \ldots, \mathcal{D}(X_n)$. The set of possible outcomes is then $\mathcal{D}(X_1) \times \cdots \times \mathcal{D}(X_n)$. For example, in the context of the problem of configuring a personal computer (PC), the variables may be *processor type, screen size, operating system* etc., where *screen size* has the domain $\{$*17in, 19in, 21in*$\}$, *operating system* has the domain {*LINUX, Windows98, WindowsXP*}, etc. Each assignment to the set of variables specifies an outcome – a particular PC configuration. Thus, a preference ordering over these outcomes specifies a partial order over possible PC configurations.

The number of possible outcomes is exponential in n, while the set of possible total orders on them is doubly exponential in n. Therefore, explicit specification and representation of a partial order are not realistic. We must find implicit means of describing this preference relation. The notion of preferential independence plays a key role in such representations. Intuitively, $\mathbf{X} \subset \mathbf{V}$ and $\mathbf{Y} = \mathbf{V} - \mathbf{X}$ are *preferentially independent* if for all assignments to $\mathbf{Y}$, our preference over $\mathbf{X}$ values are identical. More formally, let $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{D}(\mathbf{X})$ for some $\mathbf{X} \subseteq \mathbf{V}$ (where we use $\mathcal{D}(\cdot)$ to denote the domain of a set of variables as well), and let $\mathbf{y}_1, \mathbf{y}_2 \in \mathcal{D}(\mathbf{Y})$, where $\mathbf{Y} = \mathbf{V} - \mathbf{X}$. We say that $\mathbf{X}$ is *preferentially independent* of $\mathbf{Y}$ iff, for all $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{y}_1$, $\mathbf{y}_2$ we have that

$$\mathbf{x}_1\mathbf{y}_1 \succeq \mathbf{x}_2\mathbf{y}_1 \text{ iff } \mathbf{x}_1\mathbf{y}_2 \succeq \mathbf{x}_2\mathbf{y}_2$$

For example, in our PC configuration example, the user may assess *screen size* to be preferentially independent of *processor type* and *operating system*. This could be the case if, for instance, the user always prefers a larger screen to a smaller screen, no matter what the processor or the OS are.

Preferential independence is a strong property, and therefore, less common. A more refined notion is that of conditional preferential independence. Intuitively, $\mathbf{X}$ and $\mathbf{Y}$ are *conditionally preferentially independent* given $\mathbf{Z}$ if for every fixed assignment to $\mathbf{Z}$, the ranking of $\mathbf{X}$ values is independent of the value of $\mathbf{Y}$. Formally, let $\mathbf{X}, \mathbf{Y}$, and $\mathbf{Z}$ be a partition of $\mathbf{V}$ and let $\mathbf{z} \in \mathcal{D}(\mathbf{Z})$. $\mathbf{X}$ and $\mathbf{Y}$ are *conditionally preferentially independent* given $\mathbf{z}$ iff, for all $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{y}_1$, $\mathbf{y}_2$ we have that

$$\mathbf{x}_1\mathbf{y}_1\mathbf{z} \succeq \mathbf{x}_2\mathbf{y}_1\mathbf{z} \text{ iff } \mathbf{x}_1\mathbf{y}_2\mathbf{z} \succeq \mathbf{x}_2\mathbf{y}_2\mathbf{z}$$

$\mathbf{X}$ and $\mathbf{Y}$ are conditionally preferentially independent given $\mathbf{Z}$ if they are conditionally preferentially independent given any assignment $\mathbf{z} \in \mathcal{D}(\mathbf{Z})$. Returning to our PC example, the user may assess *operating system* to be independent of other features given *processor type*. That is, it always prefers LINUX given an AMD processor and Windows98 given an Intel processor (e.g., because he believes Windows98 is optimized for the Intel processor, whereas LINUX is otherwise better). Conditional preferential independence is a standard notion in multi-attribute utility theory [19].

2.2 CP-nets

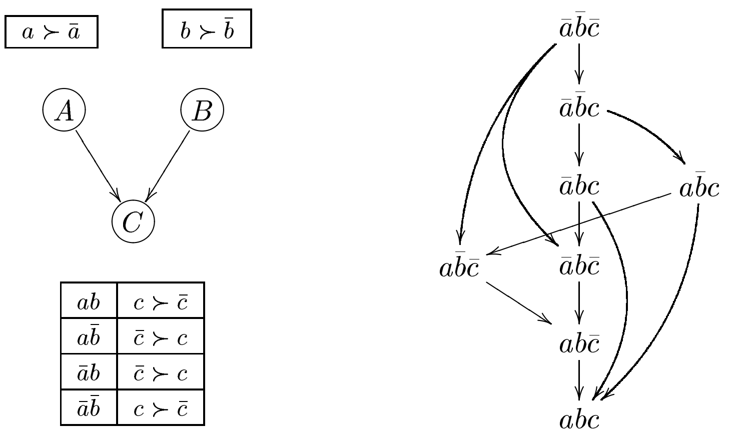
CP-nets were introduced in [3] as a tool for compactly and intuitively representing qualitative preference relations. This graphical model exploits conditional preferential independence in structuring a decision maker's preferences under *ceteris paribus* (all else being equal) semantics. CP-net is the first graphical model based on the notions of purely qualitative preferential independence, and bears a surface similarity to Bayesian nets [22]. However, the nature of the relationship between nodes within a CP-net is generally quite weak compared with the probabilistic relations in Bayes nets.

During preference elicitation, for each variable X in the variable set $\mathbf{V}$, the decision maker is asked to specify a set of *parent* variables $Pa(X)$ that can affect her preferences over the values of X. That is, given a particular value assignment to $Pa(X)$, the decision maker should be able to determine a preference order over the domain of X (denoted as $\mathcal{D}(X)$), all other things being equal.

The above information is used to create the graph of the CP-net in which each node X has $Pa(X)$ as its immediate predecessors. Given this structural information, the decision maker is asked to explicitly specify her preferences over the values of X for each instantiation of $Pa(X)$. This conditional preference over the values of X is captured by a *conditional preference table* (CPT) which is annotated with the node X in the CP-net. That is, for each assignment to $Pa(X)$, $CPT(X)$ specifies a total order (denoted $\succ$) over $\mathcal{D}(X)$, such that for any two values $x_i, x_j \in \mathcal{D}(X)$, either $x_i \succ x_j$, or $x_j \succ x_i$. Note that if the same total order is associated with a number of assignments, a representation of the CPT that is more compact than one enumerating all assignments is possible. For instance, we can associate total orders with boolean formulas over the parent variables.

Formally, a *CP-net* $\mathcal{N}$ over variables $\mathbf{V} = \{X_1, \dots, X_n\}$ is a directed graph G over $X_1, \dots, X_n$ and a set of conditional preference tables $CPT(X_i)$ for each $X_i \in \mathbf{V}$. Each conditional preference table $CPT(X_i)$ associates a total order $\succ_j (\mathbf{u}_j)$ over the domain of X_i with each instantiation $\mathbf{u}_j$ of X_i's parents $Pa(X_i) = \mathbf{U}$.

The semantics of CP-nets is defined as follows. A *model* for a CP-net is a total order over the set of possible outcomes that satisfies every CPT within the CP-net. Intuitively, a total order satisfies CPT(X) if it orders every two outcomes o, o' that differ only in the value of X appropriately. That is, it prefers the outcome in which X is assigned a better value. Recall that this is precisely the information specified in CPT(X), and that whether one



(a) (b)

Figure 1: A CP-net and its corresponding preferential order.

value of X is better than another value of X depends on the values assigned to the parents of X. But since o and o' differ only in the value of X, they agree on the value of $Pa(X)$, and this is well defined.

Formally, a total order satisfies CPT(X) if for every pair of X-values x_i, x_j and every assignment c to $Pa(X)$ such that $x_i \succ x_j$ given c according to CPT(X), the following holds: Let o, o' be two outcomes that differ only in their assignment to X and satisfy c, and suppose that o assigns x_i to X and o' assigns x_j to X, then, $o \succ o'$ holds in the model. We say that $o \succ o'$ is *valid* according to the CP-network if $o \succ o'$ holds in all models of the CP-net. A CP-net is said to be *consistent* if it has a model.

The CP-net semantics implies the following set of conditional independence assumptions: every node is conditionally independent of all variables other than its parents, given the value of its parents.

As an example, consider the network in Figure 1(a), in which all variables are boolean. This network has three variables, A, B, and C. The preferences over A and B are unconditional, whereas the preferences for C's values depend on the values of A and B. In Figure 1(b), we see the preference order induced by this CP-network. For every pair of outcomes o, o', we have that $o \succ o'$ is valid iff there is a path from o' to o in this graph.

3 The Weak Preference Semantics

According to the standard semantics, the preferences expressed in the CPT of a CP-net are *strict* preferences. We suggest viewing them as *weak* preferences using pre-orders instead of strict orders. As we show, this leads to a semantics that is only slightly weaker than the standard semantics. In fact, it leads to identical orderings on CP-nets that are consistent according to the standard semantics. More formally, we define a *model* for a CP-network as a total *pre*-order over the set of outcomes that satisfies every CPT within the CP-net. A total pre-order satisfies CPT(X) if for every pair of X-values x_i, x_j and every assignment c to $Pa(X)$ such that $x_i \succeq x_j$ given c according to CPT(X)[1] the following holds: Let o, o' be two outcomes that satisfy c and differ only in their assignment to X, such that o assigns x_i to X and o' assigns x_j to X. Then, $o \succeq o'$ holds in the ordering. We say that $o \succeq o'$ is *valid* according to the CP-network if $o \succeq o'$ holds in all models of the CP-net. Finally, we say that $o \succ o'$ is *valid* according to the CP-network if $o \succeq o'$ is valid but $o' \succeq o$ is not valid. Thus, if $o \succeq o'$ holds in all models and there is a model in which $o \succ o'$ holds, we say that $o \succ o'$ is valid. Had we required $o \succ o'$ to hold in all models for $o \succ o'$ to be valid, no strict preference would be valid. This is because the preference relation in which all outcomes are equally desirable is a model for every CP-network according to the above semantics. Thus, *all* CP-nets are "consistent" according to our semantics.

The above semantics can be related to a more syntactic notion of a proof of preference, or a flipping sequence [3]. Let o be an outcome, and suppose that o assigns to X some value x_j that the user ranks lower than x_i in parental context c satisfied by o. Then, we can improve o by flipping the value of X from x_j to x_i. An improving *flipping sequence* $o_1, o_2, \ldots, o_k$ is a sequence of outcomes such that o_i is obtained from o_{i-1} via a single improving flip. Naturally, these definitions are with respect to some CP-network N, which we take to be fixed by the appropriate context.

Theorem 1 *$o \succeq o'$ is valid (with respect to a CP-net N) iff there is an improving flipping sequence from o' to o (with respect to N).*

Proof: First, suppose that there is an improving flipping sequence from o' to o. If this sequence consists of o and o' alone, then o and o' differ in the value of a single variable, say X. This means that the CPT for X specifies

[1] In the standard semantics we would write $x_i \succ x_j$ instead of $x_i \succeq x_j$.

that X has a better value in o than in o' Thus $o \succeq o'$ must hold because every model must satisfy every CPT. For longer flipping sequences a simple inductive argument based on the transitivity property of a pre-order yields the desired result.

The other direction will follow if we can show that if there is no improving flipping sequence from o' to o then there exists a model of the CP-net in which o' is ranked higher than o.

To describe our model, we must define a total pre-order over the set of outcomes, i.e., a ranking of the outcomes. We proceed as follows: Let O denote the set of all outcomes. Let O' contain o' and the set of outcomes reachable via an improving flipping sequence from o'. Let O'' denote all other outcomes. Note that according to our assumption $o \in O''$. First, we provide an inductive definition of the rank of the elements of O''. The lowest rank (denoted by 0) contains every outcome $o_1 \in O''$ such that for every $o_2 \in O''$ such that o_1 is reachable (via an improving flipping sequence) from o_2, we also have that o_2 is reachable from o_1. Notice that this rank cannot be empty: any outcome $\bar{o}$ that is not reachable from any other outcome belongs to rank 0. If every outcome is reachable from some other outcome, there must be at least two outcomes in this rank. Rank k contains all outcomes o_1 such that $k-1$ is the maximal rank of any item o_2 such that o_1 is reachable from o_2 but o_2 is *not* reachable from o_1.

Next, we rank the elements of O': Let K denote the highest rank of an element in O''. Rank the elements of O' using the algorithm used above for O'', but with O'' replaced by O' and starting at rank $K+1$ instead of 0. By construction we get that the rank of o' is higher than the rank of o.

It remains to be shown that the ranking we constructed is indeed a model of the CP-net. First recall that a ranking defines a total pre-order as follows: $o \succeq o'$ iff o is ranked at least as high as o'. Next, suppose that o_1 and o_2 are two outcomes that differ only on some variable X, and o_1 assigns X a more preferred value than that assigned by o_2 according to the CPT for X. This means that there is an improving flipping sequence from o_2 to o_1. There are two cases to consider. If both o_1 and o_2 belong to O' or O'', our construction guarantees that o_1 will be ranked at least as high as o_2. Otherwise, it must be the case that $o_2 \in O''$ and o_1 in O' (if $o_2 \in O'$ then o_2 is reachable from o' and thus o_1 is reachable from o' as well). Thus, again, by construction o_1 is ranked higher than o_2. ∎

An immediate consequence is the following.

Consequence 1 *$o \succ o'$ is valid iff there is an improving flipping sequence from o' to o but no improving flipping sequence from o to o'.*

Another interesting theorem relates the standard, stronger semantics, and our semantics.

Theorem 2 *Let G be a CP-network that is consistent according to the standard semantics. Then, G satisfies $o \succ o'$ according to our semantics iff G satisfies $o \succ o'$ according to the standard semantics.*

Proof: Suppose that G satisfies $o \succ o'$ according to the standard semantics. This implies that there is an improving flipping sequence from o' to o according to the standard semantics. However, this is exactly a flipping sequence according to our semantics, and it follows that $o \succeq o'$ according to our semantics. Since the CP-network is consistent according to the standard semantics, there cannot be a flipping sequence from o to o', and the result follows. The converse is immediate. ▮

This theorem shows that our semantics is a direct extension of the standard semantics. To understand it better, we will soon take a look at networks that are inconsistent according to the standard semantics.

Finally, we define the notion of optimality in the context of a CP-network. There are two possible definitions: An outcome o is said to be *strongly optimal* iff there is no other outcome o' such that $o' \succeq o$ holds. An outcome o is said to be *weakly optimal* iff there is no other outcome o' such that $o' \succ o$ holds. Thus, in the first case, o is either strictly better than any other outcome or incomparable. In the second case, there may be other outcomes that are as preferred as o, but no outcome that is strictly preferred to o. In networks that are consistent according to the standard semantics, there is a unique best outcome that is strictly better than any other outcome. However, when we move beyond this class of networks, we can have more than one optimal outcome, and it can be either strongly or weakly optimal. The latter class is computationally more challenging to identify, and so here we concentrate on strongly optimal outcomes.

4 Optimality in Cyclic Networks

The semantics of CP-nets allows for cycles. But only the consistency of acyclic CP-nets is guaranteed according to the standard semantics [3]. Moreover, such networks have a unique optimal outcome. This remains true even

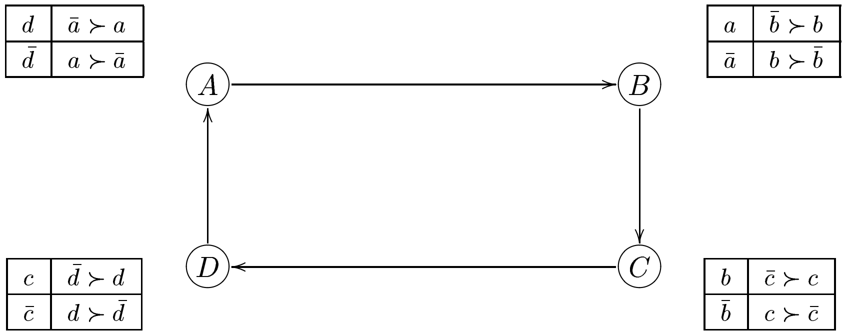


Figure 2: A Simple 4-Cycle CP-network.

if we introduce *evidential constraints*, i.e., constraints that fix the value of some variables.

Cyclic networks often induce an inconsistent (i.e., cyclic) preference order according to the standard semantics. For example, consider the network in Figure 2 which contains a cycle of size 4, called a simple cycle in [10]. The web-personalization applications described in [11] give rise to such cyclic structures. There, variables correspond to articles, ads, and other content element in an online newspaper. The value of each variable indicates whether it is currently displayed or not. The CP-net captures the preferences of the editor regarding the presentation of different elements on the user's current screen. These preferences together with user-generated content constraints lead to a personalized view that takes into account the user's interests and the editor's expertise on preferred combinations of news items. For example, in the CP-net in Figure 2, A could be a review of a new Toyota 4x4 vehicle, B a test-drive of a new BMW series 7 car, C a story about a recent Manchester United match, and D a story about Manchester City's recent success. Suppose that Manchester United is sponsored by Toyota, and Manchester City by BMW, and the editor would prefer not to display stories about competing teams and/or companies on the same screen. This is expressed in the CP-net in Figure 2 by stipulating that if A is present then B should not, if A is not present then B should be present, etc.

In [10] it is shown that such a network is not consistent, i.e., there is no total order of the set of outcomes that satisfies all the preferences embodied in this network. It is easy to see why this is so when we examine Figure 3 which describes the relationships among different possible outcomes. The

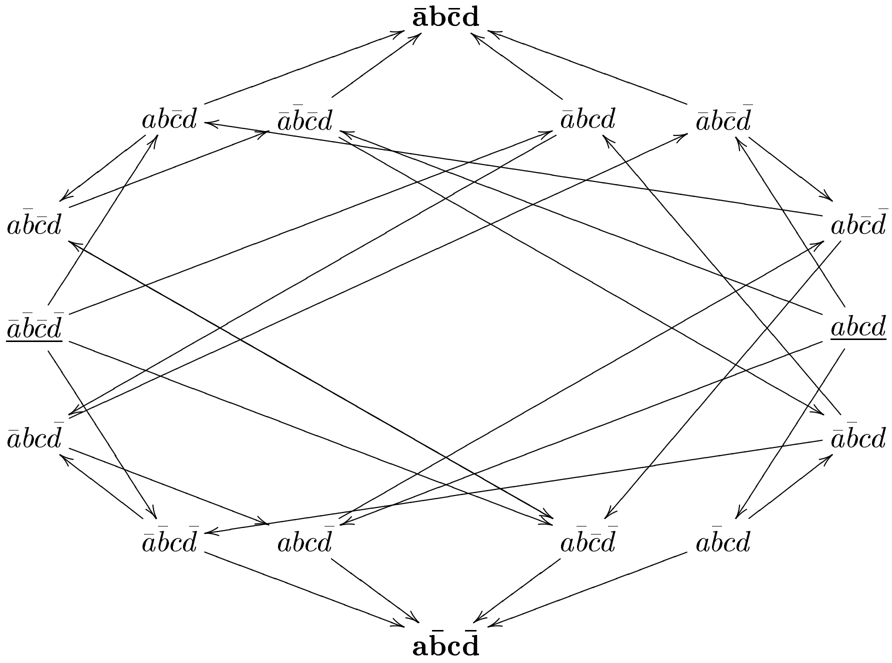


Figure 3: Outcome Space for the 4-Cycle Network.

nodes in that figure correspond to outcomes and the edges correspond to legal improving (single) flips. For example, consider the outcome $\bar{a}\bar{b}c\bar{d}$ in the lower left-hand side of Figure 3. Given that A is assigned $\bar{a}$, we can see from B's CPT that b is a more preferred value for B. Thus, there is an edge from $\bar{a}\bar{b}c\bar{d}$ to $\bar{a}bc\bar{d}$.

We can see that Figure 3 contains cycles, making it impossible to totally order its elements consistently. However, it is also apparent that it contains two elements, $a\bar{b}c\bar{d}$ and $\bar{a}b\bar{c}d$ that, in some sense, can be viewed as optimal elements, as well as two elements, $abcd$ and $\bar{a}\bar{b}\bar{c}\bar{d}$, that in some sense, can be viewed as the worst elements.

Indeed, according to our semantics, which would replace the $\succ$ relations in Figure 2 by the $\succeq$ relation, this network induces five classes of outcomes described in Figure 4. The outcomes within each class are equally preferred, and there is a strict preference between outcomes belonging to one class c over outcomes belonging to another class c' such that c is reachable from

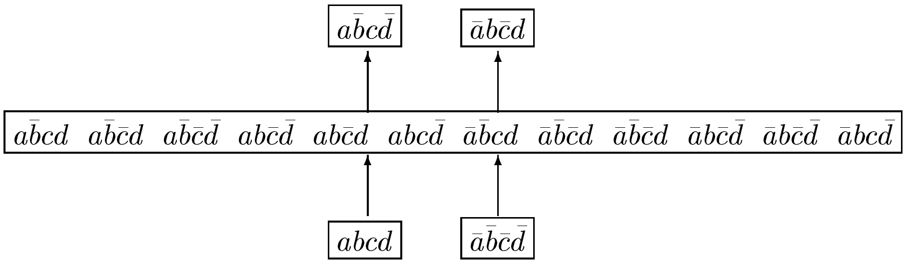


Figure 4: Ordered Outcome Classes for 4-Cycle Network

c' in this graph. An important consequence of this is that there are clear candidates for the set of optimal outcomes: the two outcomes in the two top classes.

We can see that our semantics is more lenient, allowing for specifications that, in some sense, are cyclic. If the cycle contains all outcomes, then all outcomes are equally preferred. This is not a very informative specification, and the user will have to be informed of this fact. However, as we see in the above example, the cycle may contain some, but not all of the outcomes. Whereas the standard semantics will dismiss this CP-network as inconsistent, our semantic is more tolerant, and can use this CP-net to determine an optimal outcome.

We now turn to the computation of an optimal outcome via a simple reduction to a CSP problem (or a SAT problem when the variables are binary). The variables in our reduction consist of the variables in the CP-network. For every entry in the CP Table of every variable v, we add the constraint $\phi \rightarrow v_1$, where ϕ denotes the context (i.e., the assignment to the parents of v) and v_1 is the preferred value of v.

As an example, for the CP-network in Figure 1 we obtain the following propositional formula.

$$a \wedge b \wedge (a \wedge b \rightarrow c) \wedge (a \wedge \bar{b} \rightarrow \bar{c}) \wedge (\bar{a} \wedge a \rightarrow \bar{c}) \wedge (\bar{a} \wedge \bar{b} \rightarrow c).$$

In the case of Figure 2, we get the formula

$$(d \rightarrow \bar{a}) \wedge (\bar{d} \rightarrow a) \wedge (a \rightarrow \bar{b}) \wedge (\bar{a} \rightarrow b) \wedge$$
$$(b \rightarrow \bar{c}) \wedge (\bar{b} \rightarrow c) \wedge (c \rightarrow \bar{d}) \wedge (\bar{c} \rightarrow \bar{d}).$$

Lemma 1 *o is a strongly optimal outcome for a CP-net iff it satisfies the above CSP.*

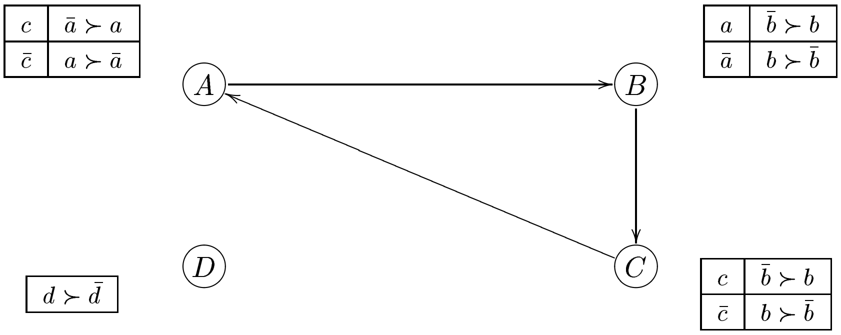


Figure 5: A Network Without a Strongly Optimal Outcome

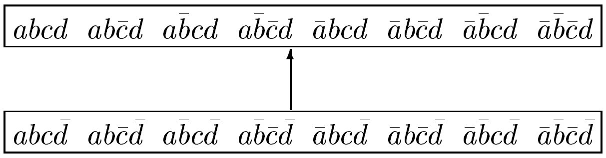


Figure 6: Outcome Classes for Network in Figure 5

Proof: If o is a satisfying assignment to the above formula, we cannot improve it via any flip. Thus, for any other outcome o', it is not the case that $o' \succeq o$, i.e., o is strongly optimal.

If o is strongly optimal then it cannot be improved via any flip. Thus, it must satisfy the above formula, which basically says that each variable is assigned the best value given the current assignment to its parents. ∎

The above algorithm does not work when we have weakly optimal outcomes, i.e., when we have a set of outcomes that are equally preferred, but cannot be strictly improved. Figures 5 and 6 show an example of a network giving rise to such a case and the relation over the outcome space it induces. From now on, our treatment centers on *strongly optimal* outcomes only, to which we simply refer as *optimal outcomes*.

Lemma 1 implies that the problem of finding a single optimal outcome is in NP. When the CP-network is acyclic, it is known that an optimal outcome can be found in linear time [3]. It is easy to understand why: If the network is acyclic, the above set of implications can be solved much like a stratified propositional logic program, and thus, be solved in linear time. Our result indicates that there is another class of networks that can be optimized in linear time: simple-cycle nets containing boolean variables. In that case, the resulting CSP becomes an instance of 2-SAT, which is solvable in linear time [1].

It is useful to relate the above optimization algorithm to the standard optimization algorithm of [3] which works on acyclic networks only. That algorithm works as follows: First, we generate a topological ordering of the variables. Next, we instantiate the value of each variable to its most preferred value given the values of its parents. Since the parents appear earlier in this ordering, this procedure is well defined. This algorithm is easily extended to the case of evidential constraints – we simply take these values to be fixed.

To see how we can generalize this procedure to cyclic networks, consider simple cycles first. Select some initial node v, and "cut" the cycle at that point. Now, for each possible value of this node, apply the optimization algorithm for acyclic networks. Finally, go back to the original network and check whether, given the resulting assignment to the parent of v, the chosen value of v is most preferred. If this is the case, the resulting outcome is strongly optimal.

As an example, consider the simple cycle in Figure 2. Suppose we select the cut-set to consist of A. The resulting network would have A as the top node, B as its only child, C as B's only child, and C's child would be D, which has no children. A can have two possible values: a and $\bar{a}$. Consider the assignment $A = a$. Optimizing the resulting acyclic network, we get $a\bar{b}c\bar{d}$. Now, we examine whether the assignment of A in this outcome is optimal with respect to the original cyclic network. Indeed, given that $D = \bar{d}$, the assignment $A = a$ is optimal. Thus, $a\bar{b}c\bar{d}$ is an optimal outcome. Similarly, when we examine the assignment $A = \bar{a}$, we generate the self-consistent outcome $\bar{a}b\bar{c}d$.

If the network is cyclic, but not a simple cycle, more than one node needs to be removed to render it acyclic. The set of removed nodes is called a *cycle cut-set*. The problem of finding a minimal cycle cut-set is better known in as the minimum vertex feedback set problem. This problem is NP-hard [14], but there are polynomial-time approximation algorithms [13], and very good heuristics (see, e.g., [2]). We perform the above process for every possible assignment to the removed nodes. An assignment obtained this way is strongly optimal iff it is *self-consistent*, i.e., the values of the removed nodes are optimal given the resulting assignment. The complexity of this procedure is linear in the number of possible assignments to the cut-set, i.e., it is exponential in the number of variables in the cut-set. Note that a self-consistent assignment corresponds to a node in the preference graph from which no other node is reachable. We formalize this procedure below:

1. Let G be the CP-network's graph. Let G' be the induced graph whose nodes are the strongly connected components of G.

2. Topologically sort G'.

3. From each strongly connected component containing more than one variable, select a cycle cut-set

4. For every assignment to the set of nodes selected in the previous step perform the standard optimization procedure under the assumption that the value of these nodes is fixed. This is called optimization in the context of evidential constraints, and it is described below.

5. Every assignment obtained in this manner which is self-consistent, is strongly optimal.

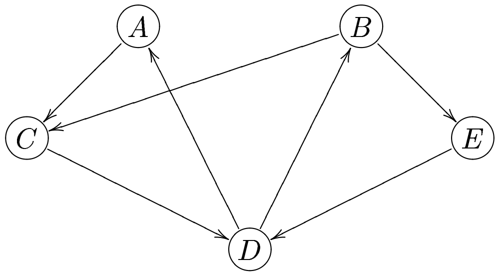
Optimization of an acyclic network given evidential constraints is done as follows: We instantiate variables according to some topological ordering. At its turn, each variable with a constrained value is assigned that value, and each unconstrained variable is assigned its optimal value given the value assigned to its parents and based on its CPT.

Lemma 2 *The set of assignments generated by the above algorithm is precisely the set of strongly optimal outcomes.*

Proof: First, every self-consistent outcome generated by the algorithm is strongly optimal by definition, as it is an outcome that cannot be improved in any way. Thus, it remains to be shown that if o is strongly optimal, it will be generated by the above algorithm. Let o_c denote the value of o on the loop cut-set variables. Since we exhaustively instantiate all values of these variables, we will consider the assignment o_c at some point. We claim when this assignment is used, our algorithm will generate o. Consider any root node n in the CP-nets obtained after cycles are removed. Its value is assigned to be optimal w.r.t. the assignment o_c. Thus, the algorithm must assign it the same value as in o, for otherwise, we could improve the value of n in o, and o would not be strongly optimal. Using an inductive argument,

d	$\bar{a} \succ a$
$\bar{d}$	$a \succ \bar{a}$

d	$b \succ \bar{b}$
$\bar{d}$	$\bar{b} \succ b$



$a \wedge \bar{b}$	$c \succ \bar{c}$
otherwise	$\bar{c} \succ c$

b	$e \succ \bar{e}$
$\bar{b}$	$\bar{e} \succ e$

$(c \wedge \bar{e}) \vee (\bar{c} \wedge e)$	$d \succ \bar{d}$
otherwise	$\bar{d} \succ d$

Figure 7: Problematic Example for Evidential Constraints

we show in this manner that the value assigned by the algorithm to all nodes must be identical to the value assigned by o. ∎

It would be nice if we could show that this procedure can be extended to deal with evidential constraints much like the algorithm of [3], i.e., by fixing the values of the observed variables. Given evidential constraints, an outcome is optimal if any outcome that is better than it according to the preference order induced by the *original* CP-net is inconsistent. This is equivalent to saying that all outcomes reachable from the given outcome in the (original) preference graph are inconsistent.[2]

Unfortunately, the cut-set algorithm no longer works in cyclic networks. To understand this, consider the CP-network in Figure 7. Suppose that C has to be assigned the value $\bar{c}$. In that case, $a\bar{b}\bar{c}\bar{d}\bar{e}$ is self-consistent in the following sense: it cannot be improved without violating the constraint, i.e., the only variable that is not assigned its most preferred value given its parents is C – which must be assigned $\bar{c}$ because of the constraint. However, we can construct a flipping sequence from $a\bar{b}\bar{c}\bar{d}\bar{e}$ to $\bar{a}b\bar{c}de$. This latter outcome is self-consistent, too, and cannot be improved with or without violating the constraint.[3]

[2]We emphasize that we consider outcome o' to be reachable from o even if the flipping sequence from o to o' includes one or more intermediate outcomes that are inconsistent.

[3]One possibility of handling evidential constraints is to alter the CP-network to reflect them, e.g., in the above network, we remove the node C and its incoming and outgoing

In general, we can say that the modified cut-set algorithm, i.e., where we generate all self-consistent assignment under the set of constraints, will result in a superset of the set of strongly optimal outcomes. This follows immediately from the fact that any strongly optimal outcome must be self-consistent, i.e., it cannot be improved without violating some constraint.

Finally, we note that for most consumer applications, we do not anticipate CP networks with more than a few dozen variables. Thus, we expect the above methods to be able to quickly return solutions in practice.

5 CP-networks and Logic Programs

We start with some background on the semantics of logic programs, followed by a reduction of CP-net optimization to logic programs. Then, using the notion of a partial stable model of a logic program, we can define a corresponding notion for a CP-net that captures those variables for which we have some reasonable candidate for an optimal value.

5.1 The Semantics of Logic Programs

A propositional *normal logic program* P is a set of *normal rules*. A normal rule is a rule of the form $h \leftarrow a_1, a_2, \ldots, a_n, not\ b_1, not\ b_2, \ldots, not\ b_m$ where $h, a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m$ are propositional atoms. The set of atoms of P is denoted by $Atoms(P)$. Let S be a truth or value assignment on the atoms of $Atoms(P)$. We denote by S^+ the set of atoms of S that are assigned the value true, and S^- the set of atoms that are assigned the value false. The *reduct* P^S of a logic program P w.r.t. an assignment S is the logic program obtained from P after deleting:

- every rule of P that has a negated atom $not\ b_i$, with $b_i \in S^+$;
- every negated atom from the body of the remaining rules.

The resulting program does not contain negated atoms, and is called a definite logic program. Let $cl(P)$ denote the deductive closure of a definite logic program P, which coincides with its minimal model. A *stable model* [15], also called an *answer set*, of a logic program P is an assignment

nodes. We alter the CPT of D, the only child of C, to reflect the fact that $C = \bar{c}$. Then we optimize the resulting network. Unfortunately, this approach violates the intended semantics of constrained optimization, e.g., in the above example, it would yield $a\bar{b}d\bar{e}$ as one of the two optimal outcomes.

S, such that $S^+ = cl(P^S)$. In the following a stable model S of a program P is represented by its subset S^+. Therefore, any element of $Atoms(P)$ that does not appear in a stable model of a program P is assumed to be false in that model.

A normal logic program may have none, one, or several stable models. The problem of deciding whether a normal logic program has a stable model is NP-complete [20], but many problems can be solved efficiently in practice by state-of-the art systems such as `smodels` [25] and `DLV` [8].

Example 1 *Consider the program*
$a \leftarrow not\ d \qquad b \leftarrow not\ a$
$c \leftarrow not\ b \qquad d \leftarrow not\ c$
This program has two stable models, namely $M_1 = \{a, c\}$ *and* $M_2 = \{b, d\}$.

Intuitively, a model is stable if using the values it assigns to each atom, we can deduce all and only the atoms to which it assigns a positive value, using the rules of the program.
In addition to normal rules, we assume that our language has *constraints*, i.e., rules with an empty head, in other words, rules of the form $\leftarrow a_1, a_2, \ldots, a_n, not$ b_1, $not\ b_2$,..., $not\ b_m$. Intuitively, such a rule states that any set of atoms that contains all of the a_i's and none of the b_j's cannot be a solution (i.e., a stable model). Formally, a constraint is a shorthand for a normal rule $f \leftarrow not\ f, a_1, a_2, \ldots, a_n, not\ b_1, not\ b_2, \ldots, not\ b_m$, where f is a new atom.

An alternative semantics for normal logic programs is the *partial stable model* semantics [23]. This is a three-valued semantics, where each atom may assume the values: true, false or undefined. In the context of a three-value semantics, a value assignment is represented by a set of literals. Given such an assignment S, S^+ denotes positive literals of S (i.e., the set of atoms of S that are assigned the value true) and S^- the negative literals of S (i.e., the set of atoms of S that are assigned the value false). An atom A such that $A \notin S^+$ and $\neg A \notin S^-$, is an undefined atom (i.e., no value is assigned to A).

Given a three-valued assignment S of $Atoms(P)$, we say that S is a *partial model* of P if for each $\neg A \in S^-$, every rule with head A contains at least one literal B in its body, such that $\neg B \in S$. The reduct P^S of P w.r.t. a partial model S is defined as the program that is obtained from P after deleting:

- every rule of P that has a negated atom $not\ b_i$, with $b_i \in S^+$

- every rule where an undefined atom occurs
- every negated atom from the body of the remaining rules

A partial model S such that $S^+ = cl(P^S)$ is called a *founded model* of P. A maximal (w.r.t. set inclusion) founded model is called *partial stable model* of P. It is not hard to see that every program has a partial stable model.

Example 2 *Consider the following program:*
$a \leftarrow not\ c \quad b \leftarrow not\ a \quad c \leftarrow not\ b$
$d \leftarrow not\ e \quad e \leftarrow not\ d$
This program has two partial stable models, namely $M_1 = \{d\}$ and $M_2 = \{e\}$, but no stable model.

The intuition behind a founded model is similar to that behind a stable model, except that we are willing to ignore certain variables and the rules that they participate in. A partial stable model is one where we try to ignore as few variables (w.r.t. set inclusion) as we can.

5.2 CP-nets Optimization Using Logic Programs

In this section we translate CP-nets into nonmonotonic logic programs, and show that there is a one-to-one correspondence between the optimal outcomes of the network and the stable models of its corresponding logic program. Naturally, the semantics of CP-nets we assume is the weak-order semantics introduced earlier in this paper.

In order to simplify the presentation of our results, and without loss of generality, we assume that the values of the variables are all distinct. This allows us to view outcomes as *sets of values*, rather than as assignments of values to variables. This considerably simplifies the proofs.

The translation for networks without constraints is rather straightforward. In fact, the translation could be obtained simply by translating the constraint satisfaction problem formulated in section 4 into a logic program, as described in [21]. We present this formulation here for the sake of completeness, and as a tool in understanding the relation of CP-nets to a large body of work on modeling preferences in logic programming. Moreover, later on, we shall discuss the extension of this translation in the context of networks with constraints – a problem of which we know of no translation to CSP.

Given a CP-network N, we construct its corresponding logic program P_N as follows. For each value x_i in the domain of a variable X of N, we introduce in P_N the atom x_i. A preference $p_1 \vee p_2 \vee \ldots \vee p_k : q_1 \succ q_2 \succ \ldots \succ q_n$, of N, where each p_i is a conjunction of values, translates into the set of rules
$q_1 \leftarrow p_1 \quad q_1 \leftarrow p_2 \quad \ldots \quad q_1 \leftarrow p_k$
Note that values other than the most preferred value, do not play any role here. Additionally, for every variable X of the network with $\mathrm{dom}(X)=\{v_1, v_2, \ldots, v_n\}$ a set of rules of the form $v_i \leftarrow not\ v_1, not\ v_2, \ldots, not\ v_{i-1}, not\ v_{i+1}, \ldots, not\ v_n$, for each $1 \leq i \leq n$, is added to the program. We call this set *choice rules set*. Finally for each variable X, with $\mathrm{dom}(X)=\{v_1, v_2, \ldots, v_n\}$ we add the set of *uniqueness constraints* $\leftarrow v_i, v_j$, for every pair of values such that $i \neq j$.

Theorem 3 *o is a strongly optimal outcome for a CP-net N iff it is a stable model of its corresponding logic program P_N.*

Proof: Assume that N is a CP-net and let S be a stable model of P_N. Then, S is a correct outcome (assigns exactly one value to every variable) since it satisfies the uniqueness constraints and choice rules set. Furthermore, (the outcome corresponding to) S cannot be improved via any flip, and therefore S is a (strongly) optimal outcome. Conversely, assume that S is an optimal outcome. Since it cannot be improved, it will satisfy all the rules of P_N. ∎

Example 3 *Consider the CP-net N:*
$a_1 : b_1 \succ b_2 \quad a_2 : b_2 \succ b_1 \quad b_1 : a_1 \succ a_2 \quad b_2 : a_2 \succ a_1$
The corresponding logic program P_N is
$b_1 \leftarrow a_1 \quad b_2 \leftarrow a_2 \quad a_1 \leftarrow b_1 \quad a_2 \leftarrow b_2$
$a_1 \leftarrow not\ a_2 \quad a_2 \leftarrow not\ a_1 \quad b_1 \leftarrow not\ b_2 \quad b_2 \leftarrow not\ b_1$
$\leftarrow a_1, a_2 \quad \leftarrow b_1, b_2.$
The program P_N has exactly two stable models, namely $M_1 = \{a_1, b_1\}$ and $M_2 = \{a_2, b_2\}$, that are exactly the optimal outcomes of N.

5.3 Partial Stable Models and Partial Outcomes

We showed a direct correspondence between the optimal outcomes of a CP-network and the stable models of its associated logic program. However, some logic programs have *no* stable model, much as some CP-nets have no strongly optimal outcomes. We can interpret the non-existence of an optimal outcome as an indication that some of the preferences are not well-defined. There are two ways to handle such situations: we can isolate the ill-defined preferences and reason with the rest of the network, or we can try to revise the ill-defined preferences. In both case, we require the ability to identify the problematic parts of the network. Here we show how partial stable models can provide valuable assistance in this task.

Example 4 *Consider the network defined as follows.*
$a_1 : b_1 \succ b_2$ $\quad a_2 : b_2 \succ b_1$ $\quad b_1 : a_2 \succ a_1$ $\quad b_2 : a_1 \succ a_2$
$c_1 : d_1 \succ d_2$ $\quad c_2 : d_2 \succ d_1$ $\quad d_1 : c_1 \succ c_2$ $\quad d_2 : c_2 \succ c_1$
Its corresponding logic program is
$b_1 \leftarrow a_1$ $\quad b_2 \leftarrow a_2$ $\quad a_2 \leftarrow b_1$ $\quad a_1 \leftarrow b_2$
$c_1 \leftarrow d_1$ $\quad c_2 \leftarrow d_2$ $\quad d_1 \leftarrow c_1$ $\quad d_2 \leftarrow c_2$
together with the corresponding choice rules set and uniqueness constraints. This program has no stable model, but it has two partial stable models, $M_1 = \{c_1, d_1\}$ *and* $M_2 = \{c_2, d_2\}$*. These partial stable models do not assign a value to variables* a_1, a_2, b_1, b_2 *because the corresponding subnetwork does not possess an optimal outcome.*

We call the outcomes of a CP-network that correspond to the partial stable models of its associated logic program, *partial optimal outcomes.*

To understand partial optimal outcomes, recall that a partial stable model corresponds to a stable model of a sub-program that is obtained by ignoring some subset of variables and all rules that mention them. Recall that our translation of CP-nets to logic programs associates a variable in the program with every variable-value in the CP-net. In what follow, we will abuse the term *value* and use it to denote variables in the LP that correspond to values of a variable in the CP-net. We claim that the following holds:

Proposition 1 *Given a partial stable model, there is a variable in the program representing the value of a CP-net variable X which is unassigned in this model iff no variable representing a value of X is assigned* true*.*

Proof: Let x_j be (a variable representing) a value of X, and suppose that x_j is unassigned in the partial stable model under consideration. Suppose to the contrary that some other value of x_i corresponds to a value of (CP-net variable) X is assigned the value *true*. In that case, we claim that the model is not maximal, since x_j can be assigned the value *false*. To see this, note that this assignment will not violate any of the choice or uniqueness rules. In addition, any rule in which x_j occurs, corresponds to a preference for some child of X in the context $X = x_j$ (where here we abused notation and use x_j to denote a value of X as well), would only involve a positive assignment to x_j, and thus would not be active if x_j is assigned *false*.

For the other direction, suppose no variable-value is assigned the value *true*. This violates our choice rules, and consequently, to inactivate them, one of the variable-value of X must be unassigned. ∎

Thus, for example, it is not possible for a partial stable model to assign *true* to two variable-values of the same CP-net variable X by making some other values of X unassigned. This seems to justify a view of partial stable model as describing partial assignment to the variables of a CP-net: Those variables with some value assigned *true* in the model can be viewed as the assigned variables, and variables for which no value is assigned can be viewed as unassigned variables.

Existing knowledge on partial stable models (e.g., [9]) implies that a total partial optimal outcome (i.e., one that assigns a value to every variable) is an optimal outcome, and that deciding whether a logic program has a partial stable model other than the empty set is NP-complete. Discovering whether a variable appears in *some* partial optimal outcome is NP-complete, whereas deciding whether a variable appears in *all* partial optimal outcomes is Π_2^p-complete.

Since problems in NP can be typically dealt with efficiently, this suggests that we can quickly supply the user with useful information about consistency. We can choose one of two options: generate some partial stable model and inform the user about variables that do not appear in that model, or check which variables are not assigned in any partial stable model.

6 Stable Models and Constraints

CP-net optimization is much more interesting and challenging given constraints on the set of outcomes. In this section we try to understand some properties of stable models in this context. First, we consider what can be obtained using the basic idea of a stable model in the case of outcome optimization. Then, we consider a standard reasoning problem for nonmonotonic formalisms: whether a particular partial assignment is credulously implied by a CP-network.

6.1 Optimization Given Constraints

The semantic notions behind stable models and self-consistent outcomes are closely related. In both cases, we look for an assignment that will, in some sense, support itself. Moreover, we would like these models to be optimal – i.e., they cannot be improved. This is reminiscent of the notion of a fix-point.

When there are no constraints, only strongly optimal outcomes cannot be improved. However, when constraints are introduced there can be non-optimal outcomes that cannot be locally improved without violating some constraint. Such outcomes can be viewed as "local maxima". This was illustrated in the case of consistent outcomes for the network of Figure 7. It appears that stable models correspond to such local maxima, rather than to global maxima. Recall that the problem of computing a stable model is in NP. Below we show that the problem of checking the optimality of an outcome is coNP-hard. This indicates that (unless NP=coNP) global optimization probably cannot be captured by the stable models of a polynomial-sized logic program induced by a CP-net.

Theorem 4 *Let O be an outcome of a CP-network N. Deciding whether O is an optimal outcome of N is coNP-hard.*

Proof: Let S be a 3-CNF formula. We construct a CP-network N_S such that some outcome O is an optimal outcome for N_S iff S is unsatisfiable. The variables of N_S consist of one binary variable for each atom in S and one additional binary variable s. For each clause $l_1 \vee l_2 \vee l_3$ of S we add the constraint $\neg l_1 \wedge \neg l_2 \wedge \neg l_3 \rightarrow \neg s$ to N_S. Finally, the only preference is $: s \succ \neg s$. It is easy to see that $O = \{\neg s\} \cup M$, where M is any valuation of the atoms of S, is an optimal outcome of N_S iff S is unsatisfiable. Note that this reduction assumes indifference between the values of the original variables.

A slightly more complicated reduction does not require indifference. Basically, we add a copy of s for each original variable. The above constraints are repeated for each of these new copies of s. In addition, each copy of s is a parent of one of the original variables. For each variable p and its parent s_p, the preferences are $s_p : \neg p \succ p$ and $\neg s_p : p \succ \neg p$. The outcome in which all s copies are assigned *false* and all original variables are assigned *true* is optimal iff S in unsatisfiable. Clearly, if S is unsatisfiable, then we cannot assign any s copy the value *true*. If S is satisfiable, then any assignment in which all s copies are *true* is better than the above assignment. Here we used a known fact about CP-nets, i.e., that it is always better to improve a parent node than to improve any of its children. ∎

The fact that the CP-net used in this proof is completely flat (i.e., there are no edges) implies that this problem is hard regardless of of the network's structure. However, we expect the complexity of constrained optimization given more complex structure, and in particular cyclic structures, to be much harder.

If we wish to use logic programs to generate the local optima (e.g., as candidates for strongly optimal outcomes), the translation of CP-nets into nonmonotonic logic programs used in the previous section needs to be modified, as it does not distinguish between preferences and constraints.

Given a CP-network N, we construct its corresponding logic program P_N as follows. For each variable value x_i of N, we introduce in P_N a corresponding atom x_i. We denote this set of atoms by $A_P(N)$. Furthermore, for each atom $x_i \in A_P(N)$ we introduce a variable x'_i with the intuitive meaning that x'_i is true whenever the value x_i is impossible. We denote this set of atoms by $A'_P(N)$.

A preference of the form $p_1 \vee p_2 \vee \ldots \vee p_k : q_1 \succ q_2 \succ \ldots \succ q_n$, where each p_i is a conjunction of values, translates into a set of rules of the following form

$$q_i \leftarrow p_j, q'_1, q'_2, \ldots, q'_{i-1}, not\ q'_i$$

for $1 \leq i \leq k$ and $1 \leq j \leq n$. Additionally, the choice rules set, and the uniqueness constraints are added to the program.

In order to simplify our discussion, we assume that the constraints of a CP-net are ternary. Extension of the results and discussion to more general constraints is straightforward. Let $l_{ik} \vee l_{jm} \vee l_{np}$ be such a constraint on the values of the variables v_i, v_j and v_n of N. This constraint translates to the rules:

$l'_i \leftarrow not\ l_{jm},\ not\ l_{np}$ for $l_i \in Dom(v_i) - \{l_{ik}\}$
$l'_j \leftarrow not\ l_{ik},\ not\ l_{np}$ for $l_j \in Dom(v_j) - \{l_{jm}\}$
$l'_n \leftarrow not\ l_{jm},\ not\ l_{ik}$ for $l_n \in Dom(v_n) - \{l_{np}\}$.

If S is a set of atoms $S \subseteq A_P(N) \cup A'_P(N)$ from the logic program P_N corresponding to a CP-net N, we denote by S^N the set of atoms $S^N = S \cap A_P(N)$.

Example 5 *Consider the CP-network N, defined on the variables A and B with domains $\{a_1, a_2\}$ and $\{b_1, b_2\}$ respectively, with the following preferences:*
$a_1 : b_1 \succ b_2 \qquad a_2 : b_2 \succ b_1$
$b_1 : a_1 \succ a_2 \qquad b_2 : a_2 \succ a_1$
and the constraints $a_1 \vee b_1$ and $a_2 \vee b_2$.
With the new encoding, the preferences translate into the rules
$b_1 \leftarrow a_1, not\ b'_1 \qquad b_2 \leftarrow a_1, b'_1 \qquad b_2 \leftarrow a_2, not\ b'_2 \qquad b_1 \leftarrow a_2, b'_2$
$a_1 \leftarrow b_1, not\ a'_1 \qquad a_2 \leftarrow b_1, a'_1 \qquad a_2 \leftarrow b_2, not\ a'_2 \qquad a_1 \leftarrow b_2, a'_2$
The constraints translate into the rules
$b'_2 \leftarrow not\ a_1 \qquad b'_1 \leftarrow not\ a_2 \qquad a'_2 \leftarrow not\ b_1 \qquad a'_1 \leftarrow not\ b_2$
The uniqueness constraints and choice rules set are also added to the program. The resulting program P_N has two stable models $M_1 = \{a_1, b_2, a'_2, b'_1\}$ and $M_2 = \{a_2, b_1, a'_1, b'_2\}$ such that M_1^N and M_2^N are exactly the optimal outcomes of the network.

As expected, the set of the stable models of P_N is a superset of the optimal outcomes of the CP-network N. The following example demonstrates this relation.

Example 6 *Consider the CP-network N that defines preferences on the values of variables A, B, and C with domains $\{a_1, a_2\}$, $\{b_1, b_2\}$ and $\{c_1, c_2\}$ respectively:*
$: a_1 \succ a_2 \qquad : b_1 \succ b_2 \qquad : c_1 \succ c_2$
The network contains the constraints
$a_2 \vee b_1 \qquad b_2 \vee a_1 \qquad a_2 \vee c_1 \qquad c_2 \vee a_1$
The preferences are captured by the rules
$a_1 \leftarrow not\ a'_1 \qquad a_2 \leftarrow a'_1 \qquad b_1 \leftarrow not\ b'_1 \qquad b_2 \leftarrow b'_1 \qquad c_1 \leftarrow not\ c'_1$
$c_2 \leftarrow c'_1$
The constraints translate into the rules
$b'_2 \leftarrow not\ a_2 \qquad a'_1 \leftarrow not\ b_1 \qquad c'_2 \leftarrow not\ a_2 \qquad a'_1 \leftarrow not\ c_1$
$a'_2 \leftarrow not\ b_2 \qquad b'_1 \leftarrow not\ a_1 \qquad a'_2 \leftarrow not\ c_2 \qquad c'_1 \leftarrow not\ a_1$
The uniqueness constraints and choice rules set must also added to the program. The logic program P_N has two stable models, $M = \{a_1, b_1, c_1, a'_2, b'_2, c'_2\}$ and $M' = \{a_2, b_2, c_2, a'_1, b'_1, c'_1\}$. However, only the first corresponds to an optimal outcome.

If we transform the above the program, by embedding the constraints into the preferences, we obtain the following program
$a_1 \leftarrow b_1, c_1 \qquad a_2 \leftarrow b_2 \qquad a_2 \leftarrow c_2$

$b_1 \leftarrow a_1 \qquad b_2 \leftarrow a_2 \qquad c_1 \leftarrow a_1 \qquad c_2 \leftarrow a_2$

The stable models of this program are identical to those of P_N. The above logic program corresponds to the following CP-network N', that does not contain constraints

$b_1, c_1 : a_1 \succ a_2 \qquad b_2 : a_2 \succ a_1 \qquad c_2 : a_2 \succ a_1$

$a_1 : b_1 \succ b_2 \qquad a_2 : b_2 \succ b_1 \qquad a_1 : c_1 \succ c_2 \qquad a_2 : c_2 \succ c_1$

Therefore although N and N' have a different meaning they are translated into equivalent logic programs.

The following result explains the relation between the outcomes of a CP-network N with constraints and the stable models of its corresponding logic program P_N.

Theorem 5 *Let N be a CP-network and P_N its corresponding logic program. A set of atoms M is a stable model of P_N iff M^N is an outcome such that all immediate successors of M^N are outcomes that violate some of the constraints of N.*

Proof: We first prove that if M is a stable model of P_N then all immediate successors of M^N violate some constraint of N. Assume that M is a stable model of P_N and there is some outcome M' obtained from M^N by an improving flipping of some variable V, and M' satisfies all constraints of N. Assume that v_i is the value of V in M^N and v_j its value in M'. Since flipping V from v_i to v_j is improving, there must be a preference of the form $p : v_1 \succ \ldots \succ v_j \succ \ldots \succ v_i \succ \ldots$ in N such that p is true in M^N. Therefore there will be a rule in P_N of the form $v_i \leftarrow p, v'_1, \ldots, v'_j, \ldots, not\ v'_i$. Assume that v'_j is true in M. Therefore, there must be a constraint C in N that involves value v_j such that the constraint is satisfied by M by setting v_j to false. Then the outcome M' is not correct since it violates C, a contradiction. Therefore v'_j is false in M. Then, the rule that has v_i in its head cannot be applied; therefore M is not a stable model. Hence, there can be no outcome M' as defined above.

Now assume that M is an outcome such that all immediate successors of M violate some constraint of N. Let M' be the set of atoms of P_N defined as $M' = M \cup M''$, where $M'' \subseteq A'_P(N)$ such that $v' \in M''$ iff there is some constraint of the form $l \vee x \vee y$ in N, such that l is a (different from v) value from the same domain as v (i.e., both are possible values of the same variable) and $x, y \notin M$. We shall prove that M' is a stable model of P_N. In order to do so, we show first that for an arbitrary variable V that has been assigned the value v_i in M, there is a rule that entails v_i according to the

stable model semantics. If there is a preference of the form p: $v_i \succ \ldots$, and p is true in M, then v_i is the optimal value for V and v_i will be included in every stable model that contains p and satisfies all constraints on V by setting v_i to true. Suppose that p is true in M and there is a preference of the form $p : v_1 \succ \ldots \succ v_i \succ \ldots$. Changing the value of V to any more preferred value violates some constraint of N. Therefore, there must be some constraint C in N of the form $k_1 \vee l_2 \vee v_i$ such that all variable values of C, except for v_i, are assigned false. The program P_N contains a set of rules of the form $v'_j \leftarrow not\ k_1, not\ l_2$ that assign true to all v'_j for $j \neq i$. Additionally, the rule $v_i \leftarrow p, v'_1, v'_2, \ldots, not\ v'_i$ is used by the stable model semantics to provide support for v_i.
Now consider an atom $v' \in M''$. Since there is a constraint of the form $l \vee x \vee y$ in N, and $x, y \notin M$, there must be a rule of the form $v' \leftarrow not\ x, not\ y$ in P_N that forces v' to be included in M.
Finally, it is not hard to see that any set K of atoms of P_N such that $M' \subset K$ cannot be a stable model of P_N. ∎

We shall use the term *locally optimal* outcome to refer to an outcome of a CP-net N that is generated by its corresponding program P_N. Therefore, a locally optimal outcome is one that can not be improved without violating a constraint. Furthermore, the set of *blocking atoms* of an outcome O is a subset of $A'_P(N)$ that contains the atoms that appear in the stable model of P_N that corresponds to O. Intuitively, this set, which has been used in the proof of Theorem 5, contains an atom v' if the constraints in the network and the other values in the outcome O render the value v impossible. Formally, this set, denoted by S_O, contains an atom v' iff $v' \in A'_P(N)$ and there is some constraint of the form $l \vee x \vee y$ in N, such that l is a (different from v) value from the same domain as v (i.e., both are possible values of the same variable) and $x, y \notin O$.

6.2 Reasoning

Given that stable models correspond to local maxima, to establish that a particular stable model is optimal, we need to check that it is not dominated by another stable model. One way to do this is via the idea of a *tester* program, introduced in [18]. We illustrate this in the context of credulous reasoning.

Given a CP-network N and a variable value v, we call *credulous reasoning* the problem of finding an optimal outcome of N that contains v. *Skeptical reasoning* is the problem of deciding whether v belongs to all optimal outcomes of N. Note that this is a limited form of reasoning task – we do not attempt to check whether an arbitrary formula α is satisfied by an (all) optimal outcome(s).

We start by considering credulous reasoning given the class of *flat* CP-networks. This class is not interesting by itself, but we use it to develop the more general class of acyclic networks. Cyclic networks remain a (tough) problem for future work.

6.2.1 Flat CP-networks

A flat network contains only unconditional preferences of the form : $p_1 \succ p_2 \succ \ldots \succ p_n$. We start with a basic hardness result:

Theorem 6 *Given a CP-network N and a variable value v, deciding whether there exists an optimal outcome of N that contains v is Σ_2^P-hard.*

Proof: Given a 3-CNF instance S it is Σ_2^P-complete to decide whether a literal l from S is true in some minimal model of S (i.e., a truth assignment maximizing the number variables assigned *false*). We construct a CP-network N_S such that l is contained in an optimal outcome of N_S iff it is true in some minimal model of S. The network N_S contains a set of unconditional preferences : $\neg a \succ a$ for each atom a of S, and the set of constraints of N_S is simply the 3-CNF instance S. It is easy to see that M is a minimal model of S that contains l iff M is an optimal outcome of N_S that also contains l. ∎

The significance of this result from our perspective is that it points to the need for a two-tiered approach, as we describe now. This approach adds a *tester* program on top of the current logic program. The program **tester**(P_N, M) takes as input some outcome M of a flat CP-network N, and either generates a new outcome M' that is more preferred than M, or returns failure. In the latter case, the input outcome M is an optimal outcome of N. The tester is a superset of the logic program P_N which additionally contains the following rules.

$$
\begin{gathered}
better \leftarrow m(V, D1), m'(V, D2), D2 > D1 \\
\leftarrow not\ better \\
\leftarrow m(V, D1), m'(V, D2), not\ (D2 \geq D1)
\end{gathered}
$$

This tester program generates, whenever possible, a new outcome for N that satisfies the above rules. Predicate $m(V, D)$ states that in the input model M variable V is assigned the value D, while $m'(V, D)$ declares the same information for the new model M'. The comparison predicate $>$ refers to the preference relation defined by the CP-network, while $X \geq Y$ means that either $X > Y$ or X and Y are equally preferred.

The first rule assigns true to proposition *better* iff there is some variable V that is assigned in M' a more preferred value than in M. The second rule, which is a constraint, enforces that proposition *better* must be true in the generated stable model. Finally, the third rule, which is again a constraint, derives a contradiction whenever there is some variable in M' that is assigned a less preferred value than in M. All these rules together ensure that the new model M' generated by the tester will assign a more preferred value to at least one of the variables, and a less preferred value to none of them.

The following proposition states an important property of flat networks, where $M(V)$ denotes the value of variable V in outcome M.

Proposition 2 *Let M and M' be two outcomes of a flat CP-network N. There is an improving flipping sequence from M to M' iff*

1. *For each variable V of N holds that $M'(V) \geq M(V)$*
2. *There exists some variable K for which $M'(K) > M(K)$*

Proof: By induction on the length of the sequence. ∎

The correctness of the tester is a direct consequence of the previous proposition.

Corollary 1 *An outcome M of a flat CP-network N is optimal iff* **tester**(P_N, M) *returns failure.*

The following algorithm **Credulous-Flat**(P_N, v) returns true if v is a credulous conclusion of N and false otherwise.

Credulous-Flat(P_N, v)
While there are stable models of P_N that contain v do
Compute a new such model M;
Use **tester**$(P_N, M \cap A_P(N))$ to compute an outcome that does not contain v
and is preferred over M;
if tester fails return true;

end-while
return false;

We can use the above approach to handle *skeptical* reasoning too. Suppose we need to determine whether some variable value v is true in *all* optimal outcomes of an acyclic CP-network . We first check whether N has an optimal outcome, which is equivalent to checking whether the set of constraints of the network is satisfiable. If the check fails (meaning that the constraints are unsatisfiable), the reasoning task fails as well. If the check succeeds, we check whether N has an optimal outcome that does not contain v. If the answer is negative the skeptical reasoning task succeeds, otherwise it fails.

Example 7 *Consider the flat CP-network N, defined on the variables A, B and C, with domains $\{a_1, a_2\}$, $\{b_1, b_2\}$ and $\{c_1, c_2\}$ respectively. The preferences are:*
$: a_1 \succ a_2 \quad : b_1 \succ b_2 \quad : c_1 \succ c_2$
and the constraints $a_2 \vee b_1 \vee c_1$, $a_1 \vee b_2 \vee c_1$, $a_1 \vee b_1 \vee c_2$, $a_2 \vee b_1 \vee c_2$, $a_2 \vee b_2 \vee c_2$. The preferences translate into the rules
$a_1 \leftarrow not\ a'_1 \quad a_2 \leftarrow a'_1 \quad b_1 \leftarrow not\ b'_1 \quad b_2 \leftarrow b'_1 \quad c_1 \leftarrow not\ c'_1 \quad c_2 \leftarrow c'_1$
The constraints translate into rules:

$a'_1 \leftarrow not\ b_1, not\ c_1 \quad b'_2 \leftarrow not\ a_2, not\ c_1 \quad c'_2 \leftarrow not\ a_2, not\ b_1$
$a'_2 \leftarrow not\ b_2, not\ c_1 \quad b'_1 \leftarrow not\ a_1, not\ c_1 \quad c'_2 \leftarrow not\ a_1, not\ b_2$
$a'_2 \leftarrow not\ b_1, not\ c_2 \quad b'_2 \leftarrow not\ a_1, not\ c_2 \quad c'_1 \leftarrow not\ a_1, not\ b_1$
$a'_1 \leftarrow not\ b_1, not\ c_2 \quad b'_2 \leftarrow not\ a_2, not\ c_2 \quad c'_1 \leftarrow not\ a_2, not\ b_1$
$a'_1 \leftarrow not\ b_2, not\ c_2 \quad b'_1 \leftarrow not\ a_2, not\ c_2 \quad c'_1 \leftarrow not\ a_2, not\ b_2.$

The uniqueness constraints and choice rules set are also added to the program P_N. Assume that we want to know if a_2 is a credulous conclusion of the CP-net N. The call to **Credulous-Flat**(P_N, a_2) *may lead to the computation of the stable model $M = \{a_2, b_2, c_2, a'_1,\ b'_1, c'_1\}$. The call to* **tester**(P_N, M) *succeeds and generates the stable model $M' = \{a_1, b_1, c_2,\ a'_2, b'_2, c'_1\}$ that is preferred over M. The algorithm will then loop and attempt to generate a different stable model of P_N that again must contain a_2. The attempt succeeds and the stable model $M'' = \{a_2, b_1, c_1, a'_1, b'_2, c'_2\}$ is generated. The call to* **tester**(P_N, M) *now fails (as the outcome $a_1 b_1 c_1$ violates a constraint), therefore the call to* **Credulous-Flat**(P_N, a_2) *returns true, meaning that a_2 is a credulous conclusion of P_N.*

In order to check whether a_2 is a skeptical conclusion of N, it is first ver-

ified that N has an optimal outcome, i.e. that its set of constraints is consistent. The call **Credulous-Flat**(P_N, a_1) *generates the set* $M' = \{a_1, b_1, c_2, a'_2, b'_2, c'_1\}$*, which corresponds to an optimal outcome that does not contain* a_2*, and the conclusion that* a_2 *is not a skeptical conclusion is derived.*

6.2.2 Reasoning in Acyclic CP-networks

In this section we present the algorithm **Credulous-Acyclic** that decides whether a value v is a credulous conclusion of an acyclic CP-network N, and returns an optimal outcome of N that contains v, if one exists. The basic idea is to traverse N level by level starting from the top, and use a variant of the algorithm **Credulous-Flat** at each level. Let $\langle L_1, L_2, \ldots, L_m \rangle$ be a partition of the nodes of an acyclic CP-network N obtained as follows: L_1 contains all nodes with in-degree 0. L_i contains all nodes with in-degree 0 in the graph obtained by removing all lower level nodes.

Credulous-Acyclic(P_N, v, O_{i-1}, i)
  found:=false;
  While (there are stable models of P_N that contain v) and (not found) do
    Compute a new such model M;
    Use **tester-acyclic**$(P_N, M \cap A_P(N), L_i)$ to compute an outcome that does not contain v and is preferred over M;
    if (tester-acyclic fails) and ($i < m$)
      Compute the set $M_i = \{v | v \in M$ and v is the value of some variable $V \in L_i$
      or $v = u'$ for a value u of some variable $U \in L_i\}$;
      $O_i = O_{i-1} \cup (M_i \cap A_P(N))$;
      $P_N = P_N \cup \{u \leftarrow | u \in M_i\}$;
      found:=**Credulous-Acyclic**$(P_N, v, O_i, i+1)$;
    end-if;
    if (tester-acyclic fails) and ($i = m$) then found:=true;
  end-while
  if found return true and the set O_m
  else return false;

To invoke the algorithm we set the level parameter i to 1 and the set O_0 to the empty set. The algorithm uses the procedure **tester-acyclic**(P_N, M, L_i) which differs from the **tester** of the previous section in that it compares only the values of variables that belong to the set L_i, i.e., the values of the variables of level i.

A high level description of the algorithm is as follows. It is a *backtracking* algorithm that traverses the input CP-net level by level, extending at each level a partial solution, i.e., an outcome that is optimal only w.r.t. some of the variables, to a more complete partial solution, i.e., one that assigns optimal values to more variables. More precisely, a call to **Credulous-Acyclic**(P_N, v, i) represents an attempt to extend an outcome that is optimal w.r.t. all variables that appear at levels j, for $1 \leq j \leq i-1$, to an outcome that is optimal for all variables down to level i. Indeed, after the completion of the computation at a level $i-1$, the algorithm has computed an outcome O_{i-1} that contains v, is optimal w.r.t. all variables at levels j, where $1 \leq j \leq i-1$, and it is only locally optimal w.r.t. all other variables, i.e., those at levels $j > i-1$. The recursive call to **Credulous-Acyclic**(P_N, v, i) leads to extending outcome O_{i-1} to O_i which contains v, is optimal w.r.t. all variables at levels j, where $1 \leq j \leq i$, and it is locally optimal w.r.t. all other variables, i.e., those at levels $j > i$. If the extension of O_{i-1} to O_i is not possible, the algorithm backtracks to level $i-1$ and computes some other outcome O'_{i-1}. If this is not possible the algorithms backtracks further to level $i-2$ and so on. When the algorithm successfully completes the computation at level m, the deepest level of the input CP-net, it has computed an outcome that contains v and is optimal w.r.t. all variables of the input CP-net.

The level partitioning $\langle L_1, L_2, \ldots, L_m \rangle$ of the nodes of an acyclic CP-net N, induces a *level partitioning* $\langle O_1, O_2, \ldots, O_m \rangle$ on every outcome O of N, where O_i is the set of values assigned to the variables of N that belong to L_i. Moreover, it induces a partitioning $\langle S_1, S_2, \ldots, S_m \rangle$ on the set of blocked atoms S_O, such that $u' \in S_i$ iff the value u belongs to the domain of some variable of L_i. We denote by S_O^i the set $S_1 \cup S_2 \cup \ldots \cup S_i$.

Definition 1 *Let N be an acyclic CP-net, O an outcome of N with level partitioning $\langle O_1, O_2, \ldots, O_m \rangle$. The program $P^i_{N,O}$ is defined as $P^i_{N,O} = P_N \cup \{p \leftarrow | p \in O_1 \cup \ldots \cup O_{i-1} \cup S_O^{i-1}\}$.*

The following lemma states two properties of acyclic CP-nets that will be used in the correctness proof of algorithm **Credulous-Acyclic**.

Lemma 3 *Let O and O' be two outcomes of an acyclic CP-net N such that $O' \succeq O$, and let $\langle O_1, O_2, \ldots, O_m \rangle$ and $\langle O'_1, O'_2, \ldots, O'_m \rangle$ be their level partitionings. If k is the smallest index such that some value of O'_k is preferred over some value of O_k, the following hold:*

a) $O_j = O'_j$, *for* $1 \leq j \leq k-1$.
b) If O' *is a locally optimal outcome, it is also a possible answer to the call* **tester-acyclic**$(P^k_{N,O}, M^{O,k}, L_k)$, *where* $M^{O,i}$ *is an outcome that assigns the same values as* O *to all variables* $V \in L_j$, *for* $1 \leq j \leq i$.

Proof:
a) Since $O' \succeq O$, there must be an improving flipping sequence from O to O'. Assume that there is some value v_i in O_d, with $d < k$, that is replaced by some other value v_j in O'_d. Since the sequence is improving, the value v_j must be preferred over the value v_i. Therefore, k is not the smallest index such that some value of O'_k is preferred over some value of O_k, a contradiction.
b) We show first that $O' \cup A$, where $A \subseteq A'_P(N)$, is a stable model of the program $P^k_{N,O}$. Since O' is a locally optimal outcome of N, from theorem 5 we know that $O' \cup S^m_{O'}$ is a stable model of P_N. From a) above we know that the level partitioning of O' is of the form $\langle O_1, O_2, \ldots, O_{k-1}, O'_k, O'_{k+1}, \ldots, O'_m \rangle$. From this it follows easily that $O' \cup A$ is also a stable of the program $P_N \cup \{p \leftarrow |p \in O_1 \cup \ldots \cup O_{k-1} \cup S^{k-1}_O\} = P^k_{N,O}$.
Apart from the rules of the program $P^k_{N,O}$, the acyclic-tester program contains a set of rules of the form *better* $\leftarrow m(V, D1), m'(V, D2), D2 > D1$ and a set of constraints of the form $\leftarrow m(V, D1), m'(V, D2), not\ (D2 \geq D1)$ that both refer to variables $V \in L_k$. Finally it contains the constraint $\leftarrow$ *not better*. As in the case of the **tester** for flat networks, the predicate $m(V, D)$ states that the variable $V \in L_k$ has been assigned the value D in $M^{O,k}$. We shall show that if the predicate $m'(V, D)$ is instantiated with the values assigned to the variables $V \in L_k$ by $M^{O,k}$, the proposition *better* is derived. Given that $O' \succeq O$, we know that there can not be a value of $M^{O,k}_k$ that is more preferred than some value of O'_k. Therefore the body of none of the constraints $\leftarrow m(V, D1), m'(V, D2), not\ (D2 \geq D1)$ is satisfied. Moreover, we know that there is some value of O'_k that is preferred over some value of $M^{O,k}_k$. Hence, the body of the rule *better* $\leftarrow m(V, D1), m'(V, D2), D2 > D1$ will be satisfied by some variable $V \in L_k$, and therefore the proposition *better* is derived. ∎

We now can prove the correctness of algorithm **Credulous-Acyclic**, which is formally stated in the following theorem.

Theorem 7 *If* N *is an acyclic CP-network, the procedure* **Credulous-Acyclic**$(P_N, v, \{\}, 1)$ *returns a set of variable values* O *iff* O *is an optimal outcome of* N *that contains* v.

Proof: We first prove that if the call to **Credulous-Acyclic**$(P_N, v, \{\}, 1)$ for a CP-net N returns true and a set of values O, the set O is an optimal outcome of N that contains v. Note that O contains v by construction. Moreover the partition $\langle O_1, O_2, \ldots, O_m \langle$, for O_i as computed by the algorithm, is a level partitioning of O. It suffices to prove that O is a consistent outcome and there is no other outcome, more preferred than O, that does not contain v. By construction, O satisfies the choice rules and uniqueness constraints. Therefore, O is a consistent outcome of the CP-network N. Assume that there is some other outcome O' of N such that $O' \succeq O$ holds, O' does not contain v and it is consistent. Furthermore, assume without loss of generality, that O' is an locally optimal outcome of N'. Let $\langle O'_1, O'_2, \ldots, O'_m \rangle$ be a level partitioning of O', and let k be the smallest index such that there is a variable at level k which is assigned a more preferred value in O' than in O. Consider the execution of the algorithm **Credulous-Acyclic** at its recursive call at depth k. It is not hard to see that the input logic program for this call is $P^k_{N,O}$, and the input outcome O_{k-1}. The algorithm will first compute an outcome $M^{O,k}$ and the procedure **tester-acyclic**$(P^k_{N,O}, M^{O,k}, O_{k-1}, L_k)$ will be invoked. By lemma 3 part b), we know that the call to **tester-acyclic**$(P^k_{N,O}, M^{O,k}, O_{k-1}, L_k)$ will succeed and therefore O can not be an outcome computed by the algorithm, a contradiction.

Assume that the acyclic CP-network N has an optimal outcome O that contains v. We will show that **Credulous-Acyclic**$(P_N, v, \{\}, 1)$ can return the outcome O. Let $\langle O_1, O_2, \ldots, O_m \rangle$ be the level partitioning of O. Since O is an optimal outcome of N, from theorem 5 we know that $O \cup S^m_O$ is a stable model of the program P_N. From this it follows easily that $O \cup S^m_O$ is a stable model of the program $P_N \cup \{p \leftarrow | p \in S^m_O\}$ but also of the program $P_N \cup \{p \leftarrow | p \in S^m_O\} \cup \{p \leftarrow | p \in O_1 \cup \ldots \cup O_m\} = P^m_{N,O}$. Therefore, O can be generated by **Credulous-Acyclic** provided that procedure **tester-acyclic**$(P^i_N, M^{O,i}, L_i)$ fails for all $i \leq m$. Assume that **tester-acyclic**$(P^j_N, M^{O,j}, L_j)$ succeeds, for some $j \leq m$. This means that there is an outcome O' that does not contain v and assigns some variable at level j a value that is more preferred than the value assigned to that variable by O. Therefore, there is a improving flip from O to O', hence O cannot be an optimal outcome, a contradiction. ∎

Skeptical reasoning can be solved by an algorithm similar to the one described for flat networks.

7 Related Work

There are several approaches to embedding/expressing preferences in logical languages and more specifically logic programming (see e.g., [7] for a non exhaustive review). In most of these frameworks preferences are defined on the set of rules of the program (with a few exceptions, e.g., [24]) in contrast to the CP-networks where preferences are defined on the values of the variables.

Most closely related to CP-nets is the *Answer Set Optimization* framework (ASO) [6] that builds on and extends previous work on *Logic Programs with Ordered Disjunction* [5]. Interestingly, this independent work, also attempts to create a link between logic programming and CP-nets. In this section, we take a closer look at ASO, and compare it to the work presented here.

Similarly to a CP-net with constraints, an ASO program is a pair of programs (P_{gen}, P_{pref}). The program P_{gen}, called the *generation program*, produces answer sets that represent the possible solutions, whereas the program P_{pref}, called the *preference program*, expresses the user preferences. The program P_{gen} corresponds to the constraints of a CP-net, whereas the program P_{pref} to the CP-net itself.

An ASO preference program is a set of *preference rules* of the form

$$C_1 > \ldots > C_n \leftarrow A_1, \ldots, A_k, not\ B_1, \ldots, not\ B_m$$

where A_i and B_j are literals and C_k is a *boolean combination* over the atoms of the preference program. Boolean combinations allow us to express rules of the form $a > (b \vee c) > d \leftarrow f$ meaning that in the case where f holds, b and c are equally preferred. For the purposes of this paper it is sufficient to restrict ourselves to rules that contain only atoms. The intuitive reading of such a rule is that, if the body is true, then C_1 is preferred over C_2, C_2 over C_3, etc.

The preferences expressed by the rules of the preference program, induce an order on the answer sets of the generation program. This order is defined in terms of the *satisfaction degree* of the rules in the preference program. The satisfaction degree can be any integer value greater or equal to 1, or the special value I which is used to denote the irrelevance of a rule w.r.t. to an answer set. Formally, the satisfaction degree of a rule is defined as follows.

Definition 2 *Let M be an answer set of P_{gen} of an* ASO *program (P_{gen}, P_{pref}). Then M satisfies the rule*

$$C_1 > \ldots > C_n \leftarrow A_1, \ldots, A_k, not\ B_1, \ldots, not\ B_m$$

- *to degree I if $A_j \notin M$ for some $j \in \{1 \ldots k\}$ or $B_i \in M$ for some $i \in \{1 \ldots m\}$*
- *to degree I if $A_j \in M$ for all $j \in \{1 \ldots k\}$, $B_i \notin M$ for all $i \in \{1 \ldots m\}$, and $C_l \notin M$ for all $l \in \{1 \ldots n\}$*
- *to degree j if all $A_j \in M$, no $B_i \in M$, and $j = min\{r | C_r \in M\}$*

The satisfaction degree of rule r in M is denoted by $v_M(r)$. The preorder $\geq$ is used to compare satisfaction degrees, and is defined by $1 \geq 2 \geq 3 \ldots$. Moreover, the values I and 1 are regarded as equally preferred, i.e., $1 \geq I$ and $I \geq 1$. The notation $x > y$ is used to denote that x is *strictly* better than y.
The order on satisfaction degrees is extended to the answer sets of the generation program by defining $M_1 \geq M_2$, where M_1, M_2 are answer sets, to be true iff for every rule r_i of the preference program holds that $v_{M_1}(r_i) \geq v_{M_2}(r_i)$. Moreover, $M_1 > M_2$ is true iff $M_1 \geq M_2$ is true, and for some rule r_i of the preference program it holds that $v_{M_1}(r_i) > v_{M_2}(r_i)$.
The *optimal* model of a ASO program (P_{gen}, P_{pref}) is a stable model M of P_{gen} for which there is no other stable model M' of P_{gen} such that $M' > M$.

At first sight it seems that there is an intuitive translation of CP-nets to ASO. Indeed, [6] proposes a translation of a CP-net N to an ASO program $(P_{gen}(N), P_{pref}(N))$, where the constraints of N (together with the suitable choice rules set and uniqueness constraints) are translated into the generating program $P_{gen}(N)$ in the obvious way. Moreover, each preference of N of the form $p_1 \vee \ldots \vee p_n : q_1 \succ \ldots \succ q_m$ translates into the following set of rules of the preference program $P_{pref}(N)$:

$$q_1 > \ldots > q_m \leftarrow p_1 \qquad \ldots \qquad q_1 > \ldots > q_m \leftarrow p_n$$

Although intuitively similar, the two approaches yield different results as illustrated by the following simple example.

Example 8 *Consider the following CP-network N, representing the buying preferences of John about laptop computers:*

$$: Pentium4 \succ Pentium3$$

$$Pentium4 : XComp \succ YComp \qquad Pentium3 : YComp \succ XComp$$

John prefers a Pentium-4 computer over a Pentium-3 one. Moreover, among Pentium-4 laptops brands (assume that `XComp` *and* `YComp` *are laptop computer manufacturers) he prefers buying a $XComp$ laptop over a $YComp$ one, whereas among Pentium-3 laptops he prefers a $YComp$ laptop over a $XComp$ one.*
Clearly, John would buy a Pentium-4 XComp laptop, provided that no constraint prohibits him from doing so. The CP-network N has an optimal outcome that coincides with the intended result. However, the corresponding ASO *program $(P_{gen}(N), P_{pref}(N))$, yields two optimal models, $M_1 = \{Pentium4, XComp\}$ and $M_2 = \{Pentium3, YComp\}$.*

In order to obtain a more direct correspondence with CP-nets, [6] introduces the concept of *ranked* ASO programs, which we abbreviate by $RASO$. A $RASO$ program is a sequence of the form $(P_{gen}, P^1_{pref}, \ldots, P^n_{pref})$, that instead of a single preference program, defines a sequence of pairwise disjoint preference programs P^i_{pref}. The *rank of a rule*, denoted by $rank(r)$, is the integer number j for which $r \in P^j_{pref}$. The preference relation now is defined as follows.

Definition 3 *Let $(P_{gen}, P^1_{pref}, \ldots, P^n_{pref})$ be a* RASO *program and M_1 and M_2 answer sets of P_{gen}. Then, $M_1 \geq_R M_2$ holds if for every preference rule r such that if $v_{M_1}(r) \geq v_{M_2}(r)$ does not hold, there is a rule r' such that $rank(r') \leq rank(r)$ and $v_{M_1}(r') > v_{M_2}(r')$.*

As noted in [6], a possible ordering on the preference rules is the one that is induced by the standard dependency graph $G(P)$ of an *acyclic* preference program P. The ranking is defined recursively by assigning $rank(a) = 0$ to every atom a that has no predecessors in $G(P)$, and the maximum of the ranks of the predecessors incremented by one to all other nodes. The resulting ordering is called *canonical* in [6].

If we look again at example 8 in the light of the new semantics, we note that under the canonical ordering the rule that corresponds to the preference $: Pentium4 \succ Pentium3$ is ranked higher than the other two rules and the corresponding $RASO$ program derives the intended conclusions. In fact [6] prove that there is a direct correspondence between $RASO$ program under the canonical ordering and *acyclic* CP-nets.

From the above analysis it becomes apparent that, methodologically, ASO (and more specifically $RASO$) and the logic programming translation presented here, are two lines of research that have been pursued independently, moved in different directions, and, not surprisingly, seem to converge.

The ASO approach started from a logic programming framework that has been extended to incorporate patterns of reasoning of CP-nets. On the other hand, the departure point of our work is the extension of the CP-nets to the cyclic case. The need for an effective computational realization of CP-net reasoning and the traditionally strong link between non-monotonic reasoning and preference modeling, has lead us to consider answer set programming as a possible implementation platform for CP-nets.

However, our work aims at taking further the relation between CP-nets and logic programming, by providing

- An extended semantics for *cyclic* CP-nets. Note that acyclicity is a crucial property in the definition of $RASO$ programs.

- An extended computational implementation of acyclic CP-nets in logic programming that tackles the problem of computing with $RASO$ programs, an issue that has not been addressed in [6].

8 Conclusion

The current semantics of CP-network is too conservative in its interpretation of user specification. Thus, a large network containing a small cyclic component could be rendered inconsistent. This is contrary to the aim of research on preference elicitation techniques, i.e., reducing the burden on users. This paper suggests a new semantics that supports the identification of an optimal outcome even in such cases. Moreover, using partial stable models, it provides an important tool for identifying the problematic parts of the network, allowing an automated elicitation system to provide useful feedback to a user. Then, it moves to take a closer look at the relation between optimization and reasoning in CP-networks and the stable model semantics for nonmonotonic logic programs.

Our results are limited in that they apply to unconstrained optimization and some aspects of reasoning with acyclic CP-nets. A more thorough computational analysis of this problem is required, as well as an algorithmic approach for constrained optimization given cyclic nets. These remain challenging problems for future research. We do hope, though, that for the moderate sized CP-networks currently discussed in applications, the semantic intuitions and algorithms provided in this work will be useful in practice.

Acknowledgments Thanks to the anonymous referees for their careful reading of earlier versions, and their many useful suggestions. Ronen Brafman acknowledges the support of the Paul Ivanier Center for Robotics Research and Production Management.

References

[1] B. Aspvall, M. Plass, and R. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Proc. Letters*, 1979.

[2] Ann Becker and Dan Geiger. Approximation algorithms for the loop cutset problem. In *Proc. of 10th Conference on Uncertainty in AI*, pages 60–68, 1994.

[3] C. Boutilier, R. I. Brafman, H. H. Hoos, and D. Poole. Reasoning with conditional ceteris paribus preference statements. In *UAI'99*, 1999.

[4] R. I. Brafman and D. Friedman. Presentation adaptation for rich media messages. STRIMM Consortium Working Paper, 2003.

[5] G. Brewka, I. Niemelä, and T. Syrjänen. Implementing ordered disjunction using answer set solvers for normal programs. In *JELIA'02*, 2002.

[6] G. Brewka, I. Niemelä, and M. Truszczynski. Answer set optimization. In *Proc. of IJCAI'03*, 2003.

[7] James P. Delgrande, Torsten Schaub, and Hans Tompits. A framework for compiling preferences in logic programs. *Theory and Practice of Logic Programming*, 3, 2003.

[8] T. Dell'Armi, W. Faber, G. Ielpa, C. Koch, N. Leone, S. Perri, and G. Pfeifer. System description: DLV. In *Proceedings of LPNMR-01*, 2001.

[9] Y. Dimopoulos, B. Nebel, and F. Toni. On the computational complexity of assumption-based argumentation for default reasoning. *Artificial Intelligence*, 141, 2002.

[10] C. Domshlak and R. I. Brafman. CP-Nets – reasoning and consistency testing. In *Proc. KR'02*, 2002.

[11] C. Domshlak, R. I. Brafman, and E. S. Shimony. Preference-based configuration of web page content. In *Proc. IJCAI'01*, 2001.

[12] J. Doyle and M. Wellman. Representing preferences as *ceteris paribus* comparatives. In *AAAI Spring Sym. on Decision-Theoretic Planning*, 1994.

[13] G. Even, J. Naor, S. Rao, and B. Schieber. Approximating minimum feedback sets and multi-cuts in directed graphs. In *Proc. 4th Int. Conf. on Integer Prog. and Combinatorial Optimization*, pages 14–28, 1995.

[14] M. Garey and D. Johnson. *Computers and Intractability - A Guide to the Theory of NP-completeness.* W.H. Freeman and Company, 1979.

[15] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. ICSLP-88*, 1988.

[16] E. Gudes, C. Domshlak, and N. Orlov. Remote conferencing with multimedia objects. In *Proceedings of the Second International Workshop on Multimedia Data Document Engineering (MDDE'02)*, March 2002.

[17] S. O. Hanson. What is a ceteris paribus preference. *J. of Philosophical Logic*, 25:307–332, 1996.

[18] T. Janhunen, I. Niemelä, P. Simons, and J-H. You. Unfolding partiality and disjunctions in stable model semantics. In *Proc. of the 7th Intern. Conf. on Principles of Knowledge Representation and Reasoning, KR'00*, 2000.

[19] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-offs.* Wiley, New York, 1976.

[20] V. Marek and M. Truszczynski. Autoepistemic logic. *Journal of the ACM*, 38, 1991.

[21] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25, 1999.

[22] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[23] D. Sacca and C. Zaniolo. Stable models and non-determinism in logic programs with negation. In *Proc. PODS'90*, 1990.

[24] Chiaki Sakama and Katsumi Inoue. Representing priorities in logic programs. In *Proc. of Joint International Conference and Syposium on Logic Programming, JICSLP'96*, 1996.

[25] T. Syrjänen and I. Niemelä. The Smodels system. In *Proceedings of LPNMR-01*, 2001.