11-1-2003

# Minimality of Solution Update in Conflict Resolution: An Application of Revision Programming to von Neumann-Morgenstern Approach

Inna Pivkina

Vladik Kreinovich
*University of Texas at El Paso*, vladik@utep.edu



Comments:
Technical Report: UTEP-CS-03-29
Published in *International Journal of Intelligent Systems*, 2005, Vol. 20, No. 9, pp. 939-956.

# Minimality of Solution Update in Conflict Resolution: An Application of Revision Programming to von Neumann-Morgenstern Approach

Inna Pivkina[1] and Vladik Kreinovich[2]

[1]Department of Computer Science
New Mexico State University
Las Cruces, NM 88003
ipivkina@cs.nmsu.edu

[2]Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968
vladik@cs.utep.edu

**Abstract**

In a 1944 book that started game theory (and mathematical approach to conflict resolution), von Neumann and Morgenstern proposed the notion of a *solution*. When the situation changes, the old solution is often no longer a solution, so it needs to be updated. In practical applications, it is usually desirable to keep the solution change "minimal" in some reasonable sense. We show that for a seemingly straightforward formalization of this minimality, checking whether a change is minimal is NP-hard. We also show that by representing the notion of a solution as a collection of revision rules, we can produce a reasonable notion of minimality for which there exists a feasible algorithm for checking minimality of update.

# 1 Conflict Resolution: Brief Introduction to von Neumann-Morgenstern (NM) Approach and Formulation of the Problem

## 1.1 Main ideas and resulting definitions

To assist in conflict resolution, i.e., to help several participants (*players*), with different goals, to come to an agreement, von Neumann and Morgenstern [14] invented *game theory*; for a more modern state of basic concepts, see, e.g., [5, 16].

The main idea behind their approach to conflict resolution is as follows. A normal way to resolve a conflict situation with many players is that first, several players find a compromise between their goals, so they form a *coalition*; these coalitions merge, split, etc., until we get a coalition that is sufficiently strong to impose its decision on the others (this is, e.g., the way a multi-party parliament usually works).

The main problem with this coalition formation is that sometimes it goes on and on and never seems to stop: indeed, when a powerful coalition is formed, outsiders can often ruin it by promising more to some of its minor players; thus, a new coalition is formed, etc. This long process frequently happens in multi-party parliaments.

How to stop this seemingly endless process? In real economic life, not all outputs and not all coalitions that are mathematically possible are considered: there exist legal restrictions (like anti-trust law) and ethical restrictions (like "good business practice") that represent the ideas of social justice, social acceptance, etc. Von Neumann and Morgenstern called these restrictions the "standard of behavior". So, in real-life conflict situations, we look not for an arbitrary outcome, but only for the outcome that belongs to some *a priori* fixed set $S$ of "socially acceptable" outcomes, i.e., outcomes that are in accordance with the existing "standard of behavior". This set $S$ is called a *solution*, or *NM-solution*.

For this standard of behavior to work, we must require two things:

- First, as soon as we have achieved a "socially acceptable" outcome (i.e., outcome $x$ from the set $S$), no new coalition can force a change in this decision (as long as we stay inside the social norm, i.e., inside the state $S$).
- Second, if some coalition proposes an outcome that is not socially acceptable, then there must always exist a coalition powerful enough to enforce a return to the social norm.

In this framework, conflict resolution consists of two stages:

- first, the society selects a "standard of behavior" (i.e., a NM solution $S$);
- second, the players negotiate a compromise solution within the selected set $S$.

Let us describe Neumann-Morgenstern's formalization of the idea of "standard of behavior."

Let us denote the total number of players by $n$. For simplicity, we will identify players with their ordinal numbers, and thus, identify the set of all players with a set $N = \{1, \ldots, n\}$. In these terms, a *coalition* is simply a subset $C \subseteq N$ of the set $N$ of all players.

Let us denote the set of all possible outcomes by $X$.

To formalize the notion of an NM-solution, we need to describe the enforcement abilities of different coalitions. The negotiating power of each coalition $C$ can be described by its ability, given an outcome $x$, to change it to some other outcome $y$. We will denote this ability by $x <_C y$. In these terms, the above requirements on a "standard of behavior" $S$ can be formalized as follows:

**Definition 1** *By a* conflict situation*, we mean a triple* $(N, X, \{<_C\}_{C \subseteq N})$*, where:*

$N$ *is a finite set whose elements are called* players *or* participants*;*

$X$ *is a set whose elements are called* outcomes*;*

$<_C$ *for every* coalition $C$ *(i.e., for every subset* $C \subseteq N$*), is a binary relation on the set* $X$*.*

*A set* $S \subseteq X$ *is called a* NM-solution *if it satisfies the following two conditions:*

1. *If* $x, y \in S$*, then for every coalition* $C$*,* $x \not<_C y$*.*
2. *If* $x \notin S$*, then there exists a coalition* $C$ *and an outcome* $y \in S$ *for which* $x <_C y$*.*

One can easily see that the definition of an NM-solution depends only on the *union*

$$< = \bigcup_C <_C$$

of binary relations $<_C$ that correspond to different coalitions. Thus, we can reformulate this definition as follows:

**Definition 2** *Let* $(N, X, \{<_C\}_{C \subseteq N})$ *be a conflict situation. We say that an outcome* $y$ dominates *an outcome* $x$ *(and denote it by* $x < y$*) if there exists a coalition* $C$ *for which* $x <_C y$*.*

**Definition 3** *A subset* $S \subseteq X$ *of the set of all outcomes is called a* NM-solution *if it satisfies the following two conditions:*

1. *if* $x$ *and* $y$ *are elements of* $S$*, then* $y$ *cannot dominate* $x$*;*
2. *if* $x$ *doesn't belong to* $S$*, then there exists an outcome* $y$ *belonging to* $S$ *that dominates* $x$ *(*$x < y$*).*

## 1.2 Important case: cooperative games

The most thoroughly analyzed conflict situations are so-called *cooperative games*, in which cooperation is, in principle, profitable to all players. An outcome is usually described by the gains $x_1 \geq 0, \ldots, x_n \geq 0$ (called *utilities*) of all the players. In these terms, each outcome $x \in X$ is an $n-$dimensional vector $(x_1, \ldots, x_n)$ called a *payoff vector*. The total amount of gains of all the players is bounded by the maximal amount of money that the players can gain by cooperating; this amount is usually denoted by $v(N)$. In these terms, the set of all possible outcomes is the set of all vectors $x_i$ for which $\sum x_i \leq v(N)$.

For cooperative games, the "enforcing" binary relation $<_C$ is usually described as follows: For every coalition $C$, we can determine the largest possible amount of money $v(C)$ that this coalition can gain in the hypothetical situation when all its members work together and all the others work against them. The function $v$ that assigns the value $v(C)$ to each coalition $C$ is called a *characteristic function* of the game.

The values $v(C)$ that correspond to different coalitions $C$ must satisfy the following natural requirement: If two disjoint coalitions $C$ and $C'$ join forces, they can gain at least the same amount of money as when they acted separately. Thus, $v(C \cup C') \geq v(C) + v(C')$.

In terms of a characteristic function, a coalition can force the transition from $x$ to $y$ if the following two conditions hold:

- first, when $C$ *can* gain for itself this new amount of money, i.e., when the total amount of money gained by the coalition $C$ in the outcome $y$ does not exceed $v(C)$;
- second, when all players from the coalition $C$ gain by going from $x$ to $y$ ($x_i < y_i$ for all $i \in C$), and are thus interested in this transition.

Let us describe such conflict situations formally:

**Definition 4** *Let $n$ be a positive integer and $N = \{1, \ldots, n\}$. By a* cooperative game*, we mean a function $v : 2^N \to [0, \infty)$ for which $v(C \cup C') \geq v(C) + v(C')$ for disjoint $C$ and $C'$. For each cooperative game, we can define the conflict situation $(N, X, \{<_C\})$ as follows:*

- *$X$ is the set of all $n-$dimensional vectors $x = (x_1, \ldots, x_n)$ for which $x_i \geq 0$ and $\sum x_i = v(N)$.*
- *$x <_C y$ if $\sum \{y_i \,|\, i \in C\} \leq v(C)$ and $x_i < y_i$ for all $i \in C$.*

Von Neumann and Morgenstern have shown that NM-solutions exist for many reasonable cooperative games, and have conjectured that such a solution exists for every cooperative game. It turned out later that there exist games without NM-solutions (see [10, 16]); however, for many practical conflict situations, solutions do exist.

It is worth mentioning that while from the purely mathematical viewpoint, the set $X$ of possible outcomes can be infinite, in practice, we only have a finite

number of possible outcomes. In other words, in practical applications, we can always assume that the set $X$ of possible outcomes is finite.

## 1.3 Von Neumann-Morgenstern solution: computational complexity

From the practical viewpoint, once we have a finite set $X$ and a binary relation $<$, we must be able to find an NM-solution, i.e., a set $S \subset X$ that satisfies the two conditions from the Definition 3. Since not all conflict situations have an NM-solution, it is also important to check whether, for a given conflict situation, such a solution exists at all.

It is known that the problem of checking the existence of an NM-solution is NP-complete. Indeed, we can represent the conflict situation as a graph, with outcomes as vertices and $x < y$ if and only if there is an edge from $x$ to $y$. In graph terms, an NM-solution is a *minimum independent dominating set*, or a *kernel*. The problem of checking the existence of such a kernel is NP-complete. This result was first proven in [1]; see also [3] and [17] (Problem 9.5.10).

For a general overview of complexity of different conflict resolution notions, see, e.g., [18] and references therein.

## 1.4 Solution update: why it is necessary, and why it should be minimal

In traditional applications of game theory, researchers assume that the notion of a standard of behavior is fixed. This is a reasonable assumption when we solve conflict problems independently of one another.

In real life, preferences of different groups $<_C$ change with time. As a result, the same set $S$ that, for the previous preference relation $<^{(0)}$, represented a socially enforceable "standard of behavior", stops being enforceable; so, we must update the original set $S^{(0)}$ so that the resulting set $S$ is socially enforceable (i.e., is an NM-solution) with respect to the new relation $<$. In other words, we must find a set $S$ that is a NM-solution in the sense of Definition 3.

As we have mentioned, for the same relation $<$, there are usually many different NM-solutions. Which one should we select? From the societal viewpoint, every change in "standard of behavior" is, to some extent, stressful. Change is unavoidable – the original set $S$ no longer works – but we should try our best to make this change as painless as possible. In other words, we must make sure that all the changes that we propose are indeed necessary, i.e., that the proposed change is *minimal* (in some reasonable sense).

In conflict situations, often, there are a lot of suggestions of how to make the change. It is therefore very important to check that a proposed change is indeed minimal.

This is the problem that we will solve in the present paper.

# 2 Straightforward Approach to Defining Update Minimality and its Limitations

## 2.1 Towards formulating the problem in precise terms

To describe this problem in precise terms, we must formalize what it means for a change to be minimal. A natural way to describe the overall change from the original set $S^{(0)}$ to the new set $S$ is to simply list all the changes one by one, i.e., to list all the elements that stopped being socially acceptable (i.e., elements of $S^{(0)}$ that are not in $S$) and all the elements that started being socially acceptable (i.e., elements of $S$ that are not in $S^{(0)}$). If we can get an NM-solution $S' \neq S$ by selecting only some of these changes, then, clearly, the transition from $S^{(0)}$ to $S$ is not minimal.

What does it mean "by selecting some of the changes"? In mathematical terms, the set of all the outcomes that were added to $S^{(0)}$ is the set difference $S - S^{(0)}$, and the set of all the outcomes that were deleted from $S^{(0)}$ is the set difference $S^{(0)} - S$. Similarly, for the alternative set $S'$, the set of all the outcomes that were added to $S^{(0)}$ is the set difference $S' - S^{(0)}$, and the set of all the outcomes that were deleted from $S^{(0)}$ is the set difference $S^{(0)} - S'$. Thus, if $S'$ is obtained by selecting only some of the changes, this means that $S' - S^{(0)} \subseteq S - S^{(0)}$ and $S^{(0)} - S' \subseteq S^{(0)} - S$.

*Comment.* These two conditions can be described by a single one $S^{(0)} \Delta S' \subseteq S^{(0)} \Delta S$, where $A \Delta B \stackrel{\text{def}}{=} (A - B) \cup (B - A)$ denotes the symmetric difference between the sets $A$ and $B$.

Thus, we arrive at the following definition:

**Definition 5** *Let $X$ be a finite set, let $<$ be a binary relation on $X$, and let $S^{(0)} \subset X$. We say that an NM-solution $S$ is a* minimal update *of $S^{(0)}$ if there exist no NM-solution $S' \neq S$ for which $S' - S^{(0)} \subseteq S - S^{(0)}$ and $S^{(0)} - S' \subseteq S^{(0)} - S$.*

The question becomes: given $X$, $<$, $S^{(0)}$, and an NM-solution $S$, how can we check whether $S$ is indeed a minimal update of $S^{(0)}$?

## 2.2 Problem: straightforward minimality checking is not efficient

Since the set $X$ is finite, we can, in principle, check minimality by simply checking, for all possible proper subsets of the set of all the changes, whether making only changes from this subset will still lead to an NM-solution. The problem with this straightforward approach is that it may take too long: if we denote by $c$ the number of changes between $S^{(0)}$ and $S$, then we need to check all $2^c - 1$ proper subsets of the corresponding set of changes (to be more precise, $2^c - 2$ since we know that the empty set – corresponding to no changes at all – does not work). For large $c$, performing $2^c - 2$ steps is not computationally feasible.

As we will show, this is not because we could not find a faster algorithm: the problem of checking whether a given NM-solution is a minimal update is NP-hard:

**Theorem 1** *The problem of checking whether a given NM-solution $S$ is a minimal update of a given set $S^{(0)}$ is coNP-complete.*

**Proof.** Minimality means that for every set $S' \neq S$, if $S' - S^{(0)} \subseteq S - S^{(0)}$ and $S' - S \subseteq S^{(0)} - S$, then $S'$ is not an NM-solution. Every set $S' \subseteq X = \{x_1, \ldots, x_n\}$ can be represented as a sequence of bits $b_1, \ldots, b_n$, where $b_i = 1$ if and only if $x_i \in S'$. Checking whether $S' - S^{(0)} \subseteq S - S^{(0)}$ and $S' - S \subseteq S^{(0)} - S$, and whether a set $S'$ is an NM-solution requires polynomial time. Thus, minimality can be described as $\forall x_1 \ldots \forall x_n P$ for some polynomial-time property $P$; thus, minimality property belongs to the class coNP.

To prove that checking minimality is coNP-complete – or, equivalently, that checking non-minimality is NP-complete – it is therefore sufficient to reduce a known NP-complete problem to the problem of checking non-minimality. As such a problem, we will take the above-mentioned problem of checking the existence of an NM-solution (kernel).

Let us show how to organize the desired reduction. Let $X_0 = \{x_1, \ldots, x_n\}$ be a finite set with a binary relation $<_0$. Let us extend $X_0$ by adding a new element $x_0$ that dominates everyone else and is dominated by everyone else, i.e., let us take $X = X_0 \cup \{x_0\}$ and $a < b$ if and only if one of following three situations happen:

- $a <_0 b$;
- $a \in X_0$ and $b = x_0$;
- $a = x_0$ and $b \in X_0$.

For this new conflict situation, the set $S = \{x_0\}$ is clearly an NM-solution. Let us show that for $S^{(0)} = X_0$, the set $S$ is not a minimal NM-solution if and only if the original conflict situation $(X_0, <_0)$ has an NM-solution – this will be the desired reduction.

Indeed, by definition, the fact that $S$ is not a minimal NM-solution means that there exists an NM-solution $S' \neq S$ for which $S' - S^{(0)} \subseteq S - S^{(0)}$ and $S^{(0)} - S' \subseteq S^{(0)} - S$. Let us show that both inclusions are true for every possible set $S' \subseteq X$.

- Let us start with the first inclusion. Since $S^{(0)} = X_0$ and $S = \{x_0\}$, we have $S - S^{(0)} = \{x_0\}$. For every other set $S' \subseteq X$, we have $S' - S^{(0)} \subseteq X - S^{(0)} = X - X_0 = \{x_0\}$, thus, we will always have $S' - S^{(0)} \subseteq S - S^{(0)}$.
- In our case, $S^{(0)} - S = X_0 - \{x_0\} = X_0 = S^{(0)}$, so from $S^{(0)} - S' \subseteq S^{(0)}$, we conclude that $S^{(0)} - S' \subseteq S^{(0)} - S$.

Thus, $S = \{x_0\}$ is not a minimal NM-solution if and only if the conflict situation $(X, <)$ has a different NM-solution $S'$. Let us show that such a solution $S' \neq S$ exists if and only if the original conflict situation $(X_0, <_0)$ has a solution.

Indeed, if the original conflict situation $(X_0, <_0)$ has an NM-solution $S_0 \subseteq X_0$, then this same $S_0$ is also an NM-solution for the enlarged conflict situation $(X, <)$. Indeed, the elements from $S_0$ do not dominate each other, and they dominate every outcome from $X - S_0$; since every element from $X_0$ dominates $x_0$, the set $S_0$ dominates $x_0$ as well. Thus, $S_0$ is an NM-solution for $(X, <)$.

Vice versa, let $S' \neq S$ be an NM-solution for $(X, <)$. If $x_0 \in S_0$, then, since $x_0$ dominates all elements of $X_0$ and elements of a solution cannot dominate each other, $S'$ cannot contain any element from $X_0$. Thus, in this case, $S' = \{x_0\} = S$. So, if $S' \neq S$, this means that $S'$ cannot contain $x_0$, i.e., $S'$ must be a subset of $X_0$. Since $S'$ is an NM-solution for the enlarged conflict situation, we conclude that no two elements of $S''$ dominate each other, and every element of $X - S'$ – in particular, every element of $X_0 - S'$ – is dominated by an element from $S'$. Thus, $S'$ is also an NM-solution for the original conflict situation $(X_0, <_0)$. The reduction is proven, and hence the theorem is true.

### 2.3 What we are planning to do

We have just shown that for a straightforward definition of update minimality, checking minimality is difficult. Thus, we need to provide a better definition of update minimality, a definition for which checking minimality of an update is feasible.

In this paper, we propose a definition based on the ideas of *revision programming*, a formalism developed by V. W. Marek and M. Truszczyński to describe updates in databases and knowledge bases (see, e.g., [11, 12]).

As a side effect, we describe the possibility of using similar ideas in physics: namely, to prediction problem in relativistic space-time (when space-time is non-Euclidean).

## 3 Revision Programming and Its Application to NM Approach

### 3.1 Revision programming: basic preliminary definitions

In this subsection, we present the basic definition of revision programming [11, 12].

The knowledge forming a general database or knowledge base can be described as a finite collection of atomic statements ("atoms") of the type "patient $A$ has blood pressure 180/110" or "the robot is in Room A", etc. For most databases, the set $U$ of all possible atomic statements is usually (large but) finite. This finite set is called a *Universe of discourse*, or *Universe*, for short.

Formally, we can simply say that we have a finite set $U$; its elements are called *atomic statements*, or simply *atoms*. Subsets of $U$ are called *databases*.

To describe changes in databases, we will use the following notations: $\mathbf{in}(a)$ means that the atom $a$ is a part of the database, and $\mathbf{out}(a)$ means that the atom $a$ is not a part of the database. Expressions of the form $\mathbf{in}(a)$ or $\mathbf{out}(a)$ are called *revision literals*.

For a revision literal $\alpha$ of the type $\mathbf{in}(a)$, its *dual* $\alpha^D$ is the revision literal $\mathbf{out}(a)$. Similarly, the *dual* of $\mathbf{out}(a)$ is $\mathbf{in}(a)$. A set of revision literals $L$ is *coherent* if it does not contain a pair of dual literals.

For any set of atoms $B \subseteq U$, we denote the corresponding set of revision literals by $B^c = \{\mathbf{in}(a) : a \in B\} \cup \{\mathbf{out}(a) : a \notin B\}$.

A *revision rule* is an expression of one of the following two types:

$$\mathbf{in}(a) \leftarrow \mathbf{in}(a_1), \ldots, \mathbf{in}(a_m), \mathbf{out}(b_1), \ldots, \mathbf{out}(b_n) \qquad \text{or} \qquad (1)$$

$$\mathbf{out}(a) \leftarrow \mathbf{in}(a_1), \ldots, \mathbf{in}(a_m), \mathbf{out}(b_1), \ldots, \mathbf{out}(b_n), \qquad (2)$$

where $a$, $a_i$, and $b_j$ are atoms. The revision literal on the left hand side of $\leftarrow$ is called the *head* of the rule $r$ and denoted by $head(r)$. The set of all the literals on the right hand side of $\leftarrow$ is called the *body* of the rule $r$ and denoted by $body(r)$.

Each rule has two possible interpretations.

In *declarative* interpretation, a revision rule is viewed as a constraint on the database. For instance, rule (1) imposes the following condition: if the database contains all $m$ atoms $a_1$, ..., $a_m$, and does not contain any of the $n$ elements $b_1, b_2, \ldots, b_n$, then the database must contain $a$.

In *computational* (*imperative*) interpretation, a revision rule expresses a way to enforce a constraint. Assume that all atoms $a_i$, $1 \leq i \leq m$, belong to the current database $B$, and none of the atoms $b_j$, $1 \leq j \leq n$ belongs to $B$. Then, to enforce the constraint (1), the atom $a$ must be added to the database.

Similarly, in the case of the constraint (2), the atom $a$ must be removed from the database. A collection of revision rules is also called a *revision program*.

It is worth mentioning that in the declarative interpretation, when we formulate a rule of the type (1), what we are saying, in effect, is that either $a$ is in the database $B$, or at least one of the atoms $a_i$, $1 \leq i \leq m$, is *not* in the database, or at least one $b_j$, $1 \leq j \leq n$, is *in* the database. From the declarative viewpoint, there is no difference between the rule (1) and, e.g., the rule

$$\mathbf{out}(a_1) \leftarrow \mathbf{in}(a_2), \ldots, \mathbf{in}(a_m), \mathbf{out}(a), \mathbf{out}(b_1), \ldots, \mathbf{out}(b_n) \qquad (3)$$

However, from the computational viewpoint, there is a difference between the rules (1) and (3).

Indeed, in principle, if the original database $B$ does not satisfy the rule, then we make this rule true in two possible ways: we can force the conclusion $head(r)$ of the implication $r$ to be true (as we did), or we can make the premise $body(r)$ of the rule $r$ to be false, by removing one of the atoms $a_i$ or by adding one of the atoms $b_j$. In revision programming, the rule specifies not only what constraint we want to be satisfied, but also how exactly we want this rule to be enforced. The rule (1) says that if the implication expressed by the rule is not

true, i.e., if the premise is true and $a$ is not in the database, we should add $a$ to the database. If, in a similar situation, we want to delete $a_1$ from the databases, then we should formulate the corresponding rule in the form (3).

Revision programming also enables us to describe the situations in which we allow several different ways of revising the database. For example, in the above case, if we want to allow both adding the atom $a$ and deleting the atom $a_1$, then we can describe this by adding both rules (1) and (3) to the corresponding collection of revision rules.

Let us now describe this formally. For a database $B \subseteq U$ and an atom $a \in U$, the revision literal $\mathbf{in}(a)$ is *true* if $a \in B$ and false if $a \notin B$. Similarly, the revision literal $\mathbf{out}(a)$ is *true* if $a \notin B$ and false if $a \in B$. We say that a collection $P$ of revision rules is *satisfied* by $B$ (or, alternatively, that $B$ is a *model* of $P$) if all the implications $r \in P$ are true in this interpretation.

## 3.2 Towards a new definition of a "minimal" change: the notion of a justified revision

Revision programming formalizes the notion of a "minimal" revision as a revision in which every change must be justified, so that there will be no unnecessary, unjustified changes. The main idea behind the corresponding definition of a *justified revision* (see, e.g., [11, 12]) is as follows.

- Let $I \subseteq U$ be an *initial database*, i.e., the initial set of atoms.
- Let $P$ be a *collection of revision rules*, i.e., a set of rules (of the type (1) or (2)) that the revised database must satisfy.
- Let $R \subseteq U$ be a *revised database* that satisfies all the rules from $P$.

When is $R$ a "justified" revision of $I$?

As we have mentioned, from the viewpoint of revision literals, the initial database $I$ can be described as a collection $I^c$ of statements $\mathbf{in}(a)$ corresponding to all $a \in I$ and statements $\mathbf{out}(a)$ corresponding to all atoms $a \notin I$. Similarly, the revised database $R$ can be described as a collection $R^c$ of statements $\mathbf{in}(a)$ corresponding to all $a \in R$ and statements $\mathbf{out}(a)$ corresponding to all atoms $a \notin R$. We want to define a reasonable notion of the *minimal* update. Intuitively, the word "minimal" means that many of the statements $\mathbf{in}(a)$ and $\mathbf{out}(a)$ that were initially true remain true in the revised database, in other words, that the set $R^c$ should be "close" to the set $I^c$.

Specifically, we would like to require that every difference between these sets $I^c$ and $R^c$ is justified by the rules from $P$. Once we fix the literals $\mathbf{in}(a)$ and $\mathbf{out}(a)$ that do not change – they form the set $I^c \cap R^c$ – we can then apply the rules from $P$ and deduce other revision literals. Let us describe this deduction is detail. We want to describe the set $S$ of all the revision literals that can be deduced from $I^c \cap R^c$ by using the rules $P$.

This set can be obtained by using the following natural algorithm. At each stage $k$ of this algorithm, we have a set $S_k$ that contains both the original revision

literals from $I^c \cap R^c$ and some literals that we deduced from the original ones by using the rules. At each stage of the algorithm, this set will grow until we get all the literals that can be thus deduced. We start with the set $S_0 = I^c \cap R^c$. Once we have $S_k$, we proceed as follows: For each rule $r$, we check that all the literals from the body (premise) of this rule are already derived, i.e., are already in the set $S_k$. If they are, we check that the head $head(r)$ (conclusion) of this rule is in $S_k$; if $head(r)$ is not in $S_k$, we mark this literal $head(r)$ as deducible.

- If after reviewing all the rules, it turns out that we did not mark anything, we stop and return $S_k$ as the desired set $S$ of all literals that can be deduced from the original ones by using the rules from $P$.
- Otherwise, if some new literals were marked, we add the marked (deducible) literals to the set $S_k$. The resulting enlarged set will be the set $S_{k+1}$.

We want every literal from $R^c$ to be thus justified. In other words, we want the resulting set $S$ to coincide with $R^c$.

The above algorithm for computing what can be deduced is well known in logic programming (see, e.g., [8, 9]); the resulting set $S$ is called the *least model* of the collection consisting of the rules $P$ and of the facts $I^c \cap R^c$. If we denote the least model of a collection $P$ of rules and facts by $\mathcal{L}(P)$, then we arrive at the following definition:

**Definition 6** *Let $I, R \subseteq U$ be databases, and let $P$ be the set of rules of type* (1) *and* (2)*. We say that $R$ is a $P$-*justified revision *of $I$ (or simply* justified revision*, for short) if $R^c = \mathcal{L}(P \cup (I^c \cap R^c))$.*

By definition of the least model $\mathcal{L}$, one can easily see that if $R^c$ is a justified revision, then it satisfies all the rules from the original collection $P$.

## 3.3 It is possible to efficiently check whether a given revision is justified

It is known that the above algorithm requires polynomial (actually quadratic) time. Indeed, at each stage, we either stop, or add at least one new literal from the head (conclusion) of one of the rules. Thus, the overall number of stages cannot exceed the number of rules in the collection $P$, hence it cannot exceed the length $n$ of the original description of the problem.

At each stage, we check each literal from each rule. This checking cannot take longer than the the size of the rules; thus, each stage requires $O(n)$ computational steps. Therefore, the algorithm consists of $O(n)$ stages with $O(n)$ steps each, to the total of $O(n) \cdot O(n) = O(n^2)$ computational steps.

Thus, with this new definition of update minimality, we not only get a reasonable definition of a justified revision, but we also get a polynomial-time algorithm for checking whether a given revision is indeed justified.

*Comment.* It is worth mentioning that by using a more sophisticated algorithm, we can actually compute the least model in linear time [2, 6, 19].

*Comment.* It is known [11, 12] that every justified revision $R$ is also minimal in the sense of the above straightforward definition: there exists no set $R' \neq R$ that satisfies all the rules from $P$ and for which $R'-I \subseteq R-I$ and $I-R' \subseteq I-R$.

### 3.4 Application of revision programming to NM-solutions

In order to apply the above definition to the notion of an NM-solution, we must describe this notion in the form of "revision rules" (1) and (2). The definition of the NM-solution consists of two parts:

1. if $x$ and $y$ are elements of $S$, then $y$ cannot dominate $x$;
2. if $x$ does not belong to $S$, then there exists an outcome $y$ belonging to $S$ that dominates $x$ ($x < y$).

The first condition can be described by listing, for each pair $x < y$, the rules

$$\mathbf{out}(x) \leftarrow \mathbf{in}(y);$$

$$\mathbf{out}(y) \leftarrow \mathbf{in}(x).$$

The first rule says that if $x < y$ and $x$ is an element of $S$, then $y$ cannot be an element of $S$. The second rule says that if $x < y$ and $y$ is an element of $S$, then $x$ cannot be an element of $S$.

The second condition can be described as follows. For every element $x \in X$, let $y_1, \ldots, y_m$ be the list of all elements that dominate $x$; then, we must add the rule

$$\mathbf{in}(x) \leftarrow \mathbf{out}(y_1), \ldots, \mathbf{out}(y_m)$$

and also $m$ rules

$$\mathbf{in}(y_j) \leftarrow \mathbf{out}(x), \mathbf{out}(y_1), \mathbf{out}(y_2), \ldots, \mathbf{out}(y_{j-1}), \mathbf{out}(y_{j+1}), \ldots, \mathbf{out}(y_m)$$

corresponding to $j = 1, \ldots, m$.

The first rule says that if an element $x$ is not dominated by any outcome from the set $S$, then $x$ must itself belong to $S$ – because otherwise the second condition in the definition of an NM-solution will not be satisfied. The other $m$ rules say that if $x$ doesn't belong to $S$, and of all the outcomes $y_i$ that dominate $x$, all – with the possible exception of one of them $y_j$ – do not belong to $S$, then this remaining dominating outcome $y_j$ must belong to $S$ – again, because otherwise, there will be no outcome $y$ belonging to $S$ that dominates $x$.

# 4 Revision Programming Can Be Also Used to Find NM-Solutions

In the previous section, we described how revision programming can be used to *check* whether a given NM-solution is indeed a justified revision of the previously known solution.

It is also possible to use revision programming to *find* such justified revisions. Of course, as we have mentioned, the problem of finding an NM-solution is, in general, NP-complete, so we cannot hope that the resulting algorithm will always work fast; however, for small size problems, this algorithm is usually reasonably efficient.

This algorithm is based on the fact that certain transformations (called *shifts*) preserve justified revisions [11]. For each set $W \subseteq U$, a *$W$-transformation* is defined as follows:

- For every revision literal $\alpha$ (i.e., a literal of the form $\mathbf{in}(a)$ or $\mathbf{out}(a)$), we define
$$T_W(\alpha) \stackrel{\text{def}}{=} \begin{cases} \alpha^D & \text{when } a \in W \\ \alpha & \text{when } a \notin W. \end{cases}$$
- For every set $L$ of revision literals, we define $T_W(L)$ as the result of applying the transformation $T_W$ to all the literals from this set $L$, i.e., as
$$T_W(L) \stackrel{\text{def}}{=} \{T_W(\alpha) \,|\, \alpha \in L\}.$$
- Similarly, for every set $A$ of atoms, we define $T_W(A)$ as follows:
$$T_W(A) = \{a \,|\, \mathbf{in}(a) \in T_W(A^c)\}.$$
- Finally, for a collection $P$ of revision rules, we define $T_W(P)$ as the result of applying $T_W$ to every literal in $P$.

The Shifting Theorem [11] states that for any two databases $I$ and $J$, database $R$ is a $P$-justified revision of $I$ if and only if $T_{I\Delta J}(R)$ is a $T_{I\Delta J}(P)$-justified revision of $J$.

Let $I$ be an initial database, and let $P$ be a given collection of revision rules. Then, the resulting algorithm for finding a justified revision of $I$ is as follows:

1. Apply the transformation $T_I$ to $P$ to obtain $T_I(P)$ (corresponding to the empty initial database $J = \emptyset$).
2. Convert $T_I(P)$ into a logic program with constraints by replacing revision rules of the type (1) by
$$a \leftarrow a_1, \ldots, a_m, \text{not } b_1, \ldots, \text{not } b_n \tag{4}$$
and replacing revision rules of the type (2) by constraints
$$\leftarrow a, a_1, \ldots, a_m, \text{not } b_1, \ldots, \text{not } b_n. \tag{5}$$
We will denote the resulting logic program with constraints by $lp(T_I(P))$.

3. Apply an "answer set" programming engine (e.g., `smodels` [15]) to the program $lp(T_I(P))$. These engines return a collection of sets of atoms called *answer sets*.

4. Finally, apply the transformation $T_I$ to the answer sets produced by the engine.

As shown in [11], the resulting sets will be exactly $P$-justified revisions of $I$.

Since we have shown that the problem of finding an NM-solution can be described in terms of revision rules, we can therefore use this algorithm to find NM-solutions.

# 5 Possible Applications to Physics

## 5.1 Prediction problem – one of the main problems of physics

A similar problem occurs in physics. One of the main problem of physics is the problem of predicting the future state of the world, i.e., the problem of predicting the future values of all the physical quantities. Another important problem is the problem of describing how the world came to be what it is, i.e., the problem of describing the past values of all the physical variables.

To solve both problems, physicists use physical equations. Physical equations describe how the values of the physical quantities – in particular, the values of physical fields at different spatial points – change with time. A traditional approach of mathematical physics is to start with the values of all the physical variables and fields at some moment of time $t_0$, and then use the physical equations to predict the values of these variables and fields at all possible moments of time. This approach was first explicitly formulated by Cauchy, therefore this prediction problem is usually called, in mathematical physics, a *Cauchy problem*. For many meaningful physical equations, it is know that we can take arbitrary initial conditions for $t = t_0$, and the equations will enable us to uniquely predict the values of the corresponding quantities and fields for all moments of time.

In the traditional formulation of the Cauchy problem, we know the values $f(x_1, x_2, x_3, t)$ of all physical variables $f$ at all the spatial points $x = (x_1, x_2, x_3)$ at the same moment of time $t = t_0$. In other words, we know the values of these variables at all the points from a set $\{(x, t_0) \,|\, x \in R^3\}$ – a 3-dimensional (hyper-)plane in our 4-dimensional space-time.

## 5.2 What is a Cauchy surface: a brief physical introduction

In many practical situations, instead of knowing the values $f(x_1, x_2, x_3, t)$ of physical quantities at different spatial points $x$ at the same moment of time $t_0$, we know the values of $f$ at different points $x$ at different moments of time. For example, in stellar dynamics and cosmology, we know the physical fields around

the Earth at the present moment of time $t_0$; however, as far as distant stars and galaxies are concerned, since the light takes time to travel from them to us, the only direct information that we get about the situation there is the information of how these fields looked like when the light was emitted. In other words, we know the state near the Earth now, the state at Alpha Centauri – which is 4 light years away – four years ago, we know the state of the distant stars in our Galaxy 10-30 thousands years ago, and we know the state of the quasars 10-20 billion years ago.

In all such cases, we know the values of the physical field at all the points $p = (x, t)$ from some set $S$ that is different from the plane $t = t_0$. We would like to make sure that the set $S$ still enables us to arbitrarily set up values for all the points $p \in S$ and then uniquely predict the values of all the physical fields for all other points from space-time. Not all sets $S$ have this property; for example:

- If we take the whole space-time as $S$, then we cannot arbitrarily set up values at different points $p \in S$. Indeed, once we select the past values, the physical equations will allow us to uniquely predict the future values and therefore, we cannot set these future values independently of the past ones.

- On the other extreme, if we use only a few causally unrelated space-time points as $S$, we will be able to arbitrarily set up the values of physical fields at these points, but we will not be able to determine the values at other points uniquely.

Physicists call sets that satisfy the conditions of arbitrariness and uniqueness *Cauchy surfaces*; see, e.g., [4]. In more precise physical terms, a set $S$ is called a Cauchy surface if, first, we can arbitrarily set the values for $p \in S$ (and be consistent), and second, the values at $p \in S$ uniquely determine the values for all other points.

The problem of checking whether a given set of space-time points is indeed a Cauchy surface is especially difficult – and especially important – in cosmology, when we have to take into consideration that the space-time is curved, its topology is, in general, different from the "flat" topology of the usual physical 4-dimensional space-time $R^3 \times R$.

To describe the Cauchy surfaces in the general setting, physicists use the notion of *causality* relation $\prec$ between points in space-time: for every two space-time points $a$ and $b$, the relation $a \prec b$ means that a change in $a$ can affect the observed characteristics at a point $b$. In Newtonian space-time, in which there is no limitation on how fast signals can travel, $a \prec b$ simply means that $a = (x, t)$ is in the past of $b = (y, s)$, i.e., that $t < s$. In special relativity, where there is a bound (speed of light $c$) on how fast signals can travel, the relation $a < b$ between the two points $a = (x, t)$ and $b = (y, s)$ means that not only $t < s$, but during the time between $t$ and $s$ the signal must be able to finish its travel the distance $d(x, y)$ between the corresponding spatial points $x$ and $y$. In other words, in special relativity, $a = (x, t) \prec b = (y, s)$ if and only if $s \geq t + c^{-1} \cdot d(x, y)$. In general, the causality relation can be even more complex.

## 5.3 Cauchy surface: towards a precise definition

How can we describe the above physical definition in precise mathematical terms? Based on the above physical meaning of this notion, we can say that for a set $S$ to be a Cauchy surface, this set must satisfy the following two conditions:

*First, we cannot have two points $a, b \in S$ that causally affect each other (i.e., $a \prec b$).*

Indeed, otherwise, once we set up a value of a physical field at a point $a$, this setting may affect the value of the same field at a point $b$ and therefore, we can no longer independently set values at points $a$ and $b$. So, in this case, the first ("arbitrariness") condition in the definition of a Cauchy surface is not satisfied.

*Second, for every space-time point $a$, there must exist a point $b \in S$ that is causally related with $a$, i.e., either $a \prec b$ or $b \prec a$.*

Indeed, otherwise, by definition of the causality relation, no matter what values we set up for all $b \in S$, these values will not affect the values of physical quantities at a point $a$. As a result, the setting of values for all $b \in S$ will not uniquely determine the value at $a$. So, in this case, the second ("uniqueness") condition in the definition of a Cauchy surface is not satisfied.

## 5.4 Relationship between Cauchy surface and NM-solution

As we have mentioned, in order to solve the main problems of physical prediction, we must know a Cauchy surface. It is therefore important to find a Cauchy surface based on a given causality relation.

Although this problem seems to be very far away from conflict resolution problems with which we have been dealing so far, computationally, there is a relation. Indeed, one can see that the above definition of a Cauchy surface is, in effect, a particular case of the notion of NM-solution: in Definition 3, we can take, as $X$, the set of all space-time points and as the domination relation $<$, the relation "$a \prec b$ or $b \prec a$". Therefore, from the computational viewpoint, the problem of finding a Cauchy surface is a particular case of the problem of finding a NM-solution.

Of course, from the physical viewpoint, space-time is a continuum, it has infinitely many points; however, in computations, we can only represent finitely many points. In the simplest case, these points form a rectangular grid; in move sophisticated numerical techniques, we can have more complicated finite sets of points.

Therefore, from the computational viewpoint, the problem of finding a Cauchy surface is a particular case of the problem of finding the NM-solution for a given finite set $X$ with an ordering $<$.

## 5.5 Cauchy surface updates: why they are necessary, why they should be minimal, and how to check this minimality

As we have mentioned, the problem of finding the appropriate Cauchy surface is especially important in cosmology. Cosmology is an attempt to describe the global structure of the Universe based on known observations and physical laws. In contrast to other more down-to-Earth areas of physics, in cosmology, we have much fewer observations to use. Not surprisingly, even when a group of researchers agrees on a qualitative general structure of space-time (such as a Friedmann-Lemaitre solution to a more sophisticated Mix-master universe; see, e.g., [13]), the numerical (quantitative) characteristics are only known with a large uncertainty. As a result, what we usually know is not a single space-time model, but rather a family of close models.

To analyze the physics of the Universe as a whole, it is therefore desirable to make predictions not just in one of these models, but in all of them, to make sure that the predictions we make are indeed valid for all possible models. For that, we must find a Cauchy surface in each of these models.

Since the models are close to each other, it is reasonable to expect that the corresponding Cauchy surfaces are also close to each other. In other words, once we have found a Cauchy surface $S$ for one of these models $M = (X, <)$, the Cauchy surface $S'$ for other models $M' = (X, <')$ can be viewed as updates of $S$.

From the computational viewpoint, it is desirable to re-use $M$-related computations as much as possible when performing computations with $M'$. Therefore, it is desirable to select $S'$ that is, in some sense, minimally different from the original Cauchy surface $S$.

So, before we start using $S'$, it is desirable to check whether the update from $S$ to $S'$ is indeed minimal in some reasonable sense, i.e., in other words, to check whether it is possible or not to not make some of the changes and still get a Cauchy surface.

On the surface, this problem sounds very different from the conflict-related problems that we were solving in the main part of this text, but from the purely computational viewpoint, this is exactly the problem that we dealt with:

- we have a definition of a Cauchy surface $S$ (that is, for $<'=\prec' \cup \succ'$, exactly NM-solution),
- we have the original set $S$ that is not a Cauchy surface for this relation $<'$;
- we have a new set $S'$ that is a Cauchy surface with respect to $<'$.

We must check whether the transition from $S$ to $S'$ is indeed minimal in some sense.

Since this new Cauchy-surface problem turns out to be a particular case of the problem that we already know how to formalize and solve, we can use the

same definition and algorithm to solve this new problem, i.e., to define what it means for a Cauchy surface update to be minimal and to check whether the Cauchy surface update is indeed minimal in this sense.

It is worth mentioning that the Cauchy surface problem is a particular case of the NM-solution problem, a case in which the "dominance" relation $<$ is symmetric. Because of this symmetry, the Cauchy surface problem is somewhat easier to solve than the general problem of finding the NM-solution. Indeed, as we have mentioned, the general problem of finding the NM-solution is NP-complete. In contrast, we can find a Cauchy surface in polynomial (actually cubic) time by using the following straightforward algorithm.

In this algorithm, at each stage $k$, we have a set $S_k$ with the property that no two points from this set dominate each other. At each stage of the algorithm, this set will grow until it forms the desired Cauchy surface. We start with an empty set $S_0 = \emptyset$. Once we have $S_k$, we proceed as follows: we check whether every point $a \notin S_k$ is dominated by one of the points from $S_k$.

- If all the points $a \notin S_k$ are indeed dominated, then $S_k$ is a Cauchy surface, so we stop.
- Otherwise, once we find a point $a \notin S_k$ that is not dominated, we add this point to $S_k$, and consider a new set $S_{k+1} = S_k \cup \{a\}$. It is easy to see that, due to the symmetry of the dominance relation, no two points from the new set $S_{k+1}$ dominate each other.

At each stage, we either stop, or add a new point to the original empty set. Thus, the overall number of stages cannot exceed the number of elements in the original set $X$, hence it cannot exceed the length $n$ of the original description of the graph $(X, <)$. On each stage, for each of $\le n$ points $a \notin S_k$, we must check whether it is dominated by $\le n$ points $b \in S_k$; thus, each stage requires $O(n^2)$ computational steps. Therefore, the algorithm consists of $O(n)$ stages with $O(n^2)$ steps each, to the total of $O(n) \cdot O(n^2) = O(n^3)$ computational steps.

## Acknowledgments

I.P. was partly supported by the NSF grants EIA-9810732 and EIA-0220590. V.K. was supported in part by NASA under cooperative agreement NCC5-209, by the Future Aerospace Science and Technology Program (FAST) Center for Structural Integrity of Aerospace Systems, effort sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-00-1-0365, by NSF grants EAR-0112968 and EAR-0225670, by the Army Research Laboratories grant DATM-05-02-C-0046, and by the SC2003 Minority Serving Institutions Participation Grants.

# References


[1] V. Chvatal, *On the computational complexity of finding a kernel*, Center de Recherches Mathématiques, Université de Montréal, Technical Report CRM-300, 1973.

[2] W. Dowling and J. Gallier, "Linear Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae", *Journal of Logic Programming*, 1984, Vol. 3, pp. 267–284.

[3] M. E. Garey and D. S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, Freeman, San Francisco, 1979.

[4] R. Geroch, "Domain of dependence", *Journal of Mathematical Physics*, 1970, Vol. 11, pp. 437–449.

[5] J. C. Harshanyi, "An equilibrium-point interpretation of the von Neumann-Morgenstern solution and a proposed alternative definition", In: *John von Neumann and modern economics*, Claredon Press, Oxford, 1989, pp. 162–190.

[6] A. Itai and J. Makowsky, *On the complexity of Herbrand's theorem*, Technical Report No. 243, Department of Computer Science, Israel Institute of Technology, Haifa, 1982.

[7] V. Kreinovich, "Random sets unify, explain, and aid known uncertainty methods in expert systems", in J. Goutsias, R. P. S. Mahler, and H. T. Nguyen (eds.), *Random Sets: Theory and Applications*, Springer-Verlag, N.Y., 1997, pp. 321–345.

[8] V. Lifschitz, "Foundations of logic programming", In: *Principles of Knowledge Representation*, CSLI Publications, Stanford, CA, 1996, pp. 69–127.

[9] J. Lloyd, *Foundations of Logic Programming*, Springer-Verlag, Berlin, 1984.

[10] W. F. Lucas, "The proof that a game may not have a solution", *Trans. Amer. Math. Soc.*, 1969, Vol. 136, pp. 219–229.

[11] V. W. Marek, I. Pivkina, and M. Truszczyński, "Revision programming = logic programming + constraints, *Proceedings of the 12th International Workshop on Computer Science Logic CSL'98*, Springer-Verlag Lecture Notes in Computer Science, 1999, Vol. 1584, pp. 73–89.

[12] V. W. Marek and M. Truszczyński, "Revision programming", *Theoretical Computer Science*, 1998, Vol. 190, pp. 241–277.

[13] C. W. Misner, K. S. Thorne, and J. A. Wheeler, *Gravitation*, W. H. Freeman and Company, San Francisco, 1973.

[14] J. von Neumann and O. Morgenstern, *Theory of games and economic behavior*, Princeton University Press, Princeton, NJ, 1944.

[15] I. Niemelä and P. Simons, "Efficient implementation of the well-founded and stable model semantics", In: *Proceedings of the 1996 JICSLP*, MIT Press, 1996, pp. 289–303.

[16] G. Owen, *Game theory*, Academic Press, N.Y., 1982.

[17] C. H. Papadimitriou, *Computational Complexity*, Addison Wesley, San Diego, 1994.

[18] C. H. Papadimitriou, "Algorithms, games, and the Internet", *Proceedings of ACM Symposium on Theory of Computing STOC'01*, Hersonissos, Crete, Greece, July 6–8, 2001.

[19] M. G. Scutellá, "A note on Dowling and Gallier's top-down algorithm for propositional Horn satisfiability", *Journal of Logic Programming*, 1990, Vol. 8, pp. 265–273.