ON MANIPULABILITY OF ALGORITHMS

Sviatoslav Braynov Braynov
Department of Computer Science
University of Illinois at Springfield
Springfield, IL 62703
sbray2@uis.edu

Abstract
In adversarial situations, the input data to an algorithm could be manipulated to make the algorithm produce erroneous output or to make a wrong decision. The paper presents a formal definition and a model of algorithm manipulation from a game-theoretic point of view. Algorithm manipulation is viewed as a game between a decision maker and an adversary. The decision maker runs an algorithm to make a decision, whereas the adversary manipulates the input data to his own advantage and to the disadvantage of the decision maker.

The paper also proposes a method for decision making based on manipulated input. According to the method, the decision strategy and the manipulation must be in Nash equilibrium. In other words, the decision strategy is the best response to the manipulation and vice versa, the manipulation is the best response to the decision strategy.

Keywords: data manipulation, decision making, algorithm design, adversarial learning, adversarial planning, adversarial plan recognition.

1. INTRODUCTION
It has been traditionally assumed that the input of every algorithm is a correct representation of a computational problem to be solved by the algorithm. This assumption is innocuous in situations where the input comes from a trusted source who is interested in solving the computational problem and collaborates with the algorithm. The assumption, however, does not hold in adversarial situations where the entity that provides the input data is different from the entity that uses the output data. In such situations, the adversary might misrepresent, slant, or even falsify the input to his own advantage and to the disadvantage of the party that relies on the algorithm output. Examples of such situations are abundant. For instance, the input for a virus detection algorithm is directly supplied by the virus writer in the form of a binary file. The virus writer has a strong incentive to obfuscate the binary and make it difficult for detection. Similar problems exist in anti-terrorist data mining where the input data is generated by terrorists who modify and disguise their activities in order to avoid detection. Another example is intrusion detection where intruders modify their actions in order to pass as legitimate users. Spam filters also suffer from the same problem because spam is usually
disguised as legitimate email. A common feature of all of these situations is the possibility to manipulate algorithm input and defeat the algorithm.

Although the problem has been recently recognized and discussed by research in data mining (Dalvi et al. 2004, Lowd and Meek 2005) and computer security (Barreno et al. 2006), the general understanding of the problem is still limited. In addition, the available solutions are problem specific and tailored to particular algorithms. The contribution of this paper is a general definition of algorithm manipulation that is applicable to a wide range of algorithms and problem situations, including data mining, computer security, and antiterrorist research. Algorithm manipulation is defined as a game between two parties with conflicting interests: a decision maker and an adversary. The decision maker runs an algorithm to make a decision, whereas the adversary manipulates the input data to his own advantage and to the disadvantage of the decision maker.

Another contribution of the paper is a method for decision making based on manipulated input. In many cases, manipulation is unavoidable and decision makers must know how to detect data manipulation and limit its effect on decision making. The method proposed in the paper “reverses” manipulated data to its original state and makes a decision based on the original data.

The paper is organized as follows. Section 2 introduces definitions of algorithm manipulation and discusses their advantages and disadvantages. Section 3 proposes a method for decision making based on manipulated input. The paper concludes with a brief summary of results.

2. DEFINITION OF ALGORITHM MANIPULATION

In general, algorithm manipulation involves an algorithm that solves a particular computational problem and two parties with conflicting interests: a decision maker and a manipulator. The decision maker needs to solve an instance of the computational problem in order to make a decision that affects both him and the manipulator. The decision maker feeds the problem instance into the algorithm and makes a decision based on the algorithm output. The interesting part is that the decision maker does not have complete control over the problem instance. Before the problem instance is entered into the algorithm, it could be partially or completely modified by the manipulator, leaving the decision maker uncertain about the real problem he is trying to solve. For example, the manipulator can modify the problem instance to his advantage and to the disadvantage of the decision maker. Figure 1 represents a schematic view of the main steps of algorithm manipulation.

We assume that the manipulator knows how the algorithm works. This assumption is realistic and complies with Kerckhoffs' principle (Kerckhoffs 1883) used in cryptography. According to Kerckhoffs' principle, the security of a system cannot be based on the secrecy of its internal working because sooner or later the adversary gets to know the system.
As an example of algorithm manipulation, consider a data mining algorithm that takes a trace of suspects’ actions as input and detects patterns of potential terrorist activities. If a terrorist knows the internals and the logic of the algorithm, he would be able to find the patterns the algorithm is looking for and modify his behavior in order to avoid detection. In our previous research (Braynov 2006a, Braynov 2006b) we studied different manipulative methods, such as dummy actions, decoy goals, and buffers that could be used by terrorists to obfuscate their traces and make them difficult for analysis. We also proposed a model for detecting manipulations and for identifying cells in malicious networks.

The interaction between the manipulator and the decision maker can be modeled as a two-person non-cooperative game.

**Definition 1.** The non-cooperative game between the manipulator and the decision maker is defined as a tuple \( \langle \mathcal{A}, \mathcal{P}, \mathcal{M}, \mathcal{D}, U_M, U_D \rangle \), where:

- \( \mathcal{A} \) is an algorithm that takes a problem instance, \( P \in \mathcal{P} \), as input and produces a solution as an output \( \mathcal{A}(P) \).
- \( \mathcal{P} \) is the set of all problem instances that the algorithm can take as input.
- \( \mathcal{M} \) is the set of manipulative actions available to the manipulator. When applied to a problem instance \( P \), \( P \in \mathcal{P} \), a manipulative action produces another problem instance \( P' \), \( P' \in \mathcal{P} \). In other words, \( \mathcal{M} : \mathcal{P} \rightarrow \mathcal{P} \).
- \( \mathcal{D} \) is the decision function of the decision maker. The function produces a decision \( \mathcal{D}(\mathcal{A}(P)) \) on the output \( \mathcal{A}(P) \) produced by the algorithm for a problem instance \( P \).
- \( U_M \) is the utility function of the manipulator, \( U_M : \mathcal{D} \times \mathcal{P} \rightarrow \mathcal{R} \). It maps a pair of a decision and a problem instance into a real number.
- \( U_D \) is the utility function of the manipulator, \( U_D : D \times P \rightarrow R \). It maps a pair of a decision and a problem instance into a real number.

**Definition 2 (strong version)** An algorithm \( A \) is non-manipulable if for every manipulative action \( m, m \in M \), and every problem instance \( P, P \in P \):

\[
A(P) = A(m(P))
\]

Definition 2 says that for every manipulative action \( m \) that changes the problem instance from \( P \) to \( m(P) \), the algorithm produces the same output as with the original problem instance \( P \). In other words, a non-manipulable algorithm is not affected by manipulations and it always produces correct output, whether it works on a manipulated input or not.

It has to be pointed out that not every manipulation produces a valid problem instance. Some manipulations could corrupt the problem instance and produce illegitimate input. Such manipulations are of no practical interest because they could easily be detected and rejected using proper input validation. Throughout this paper, we assume that manipulations always produce valid problem instances. In other words, they convert one problem instance into another.

Definition 2 requires that the algorithm be somewhat insensitive to variations in problem instances and produce the same output for a large number of different inputs. This is an unrealistically strong assumption that could constrain the applicability of the model. We relax the assumption in the next definition.

**Definition 3 (weaker version)** An algorithm \( A \) is non-manipulable if for every manipulative action \( m, m \in M \), and every problem instance \( P, P \in P \):

\[
D(A(P)) = D(A(m(P)))
\]

Unlike Definition 2, Definition 3 allows the manipulator to affect the output of the algorithm. For example, by changing the problem instance from \( P \) to \( m(P) \), the attacker could make the algorithm produce output \( A(m(P)) \) that is different from \( A(P) \), the output the algorithm would produce if it was given an unmanipulated problem instance:

\[
A(P) \neq A(m(P))
\]

The interesting part about Definition 3 is that the difference in the algorithm output does not affect the decision of the decision maker. The decision remains the same on both unmanipulated, \( A(P) \), and manipulated output, \( A(m(P)) \). In other words, it is not the algorithm that is not affected by manipulations, but the decision function \( D \) of the decision maker. The intuition behind Definition 3 is that small changes in the algorithm output might not affect a decision. For example, a terrorist might try to manipulate the output of a data mining algorithm that detects patterns of terrorist activity by camouflaging and hiding some actions. As a result, the confidence level of the detected pattern could fall by 2% which, however, might not be sufficient for the decision maker to consider the pattern inconclusive of evidence. In many cases, the decision maker will allow for some variations in the algorithm output and will not change his decision until
the variations exceed some critical level. Therefore, the goal of the manipulator would be to convert the original problem instance $P$ to a new instance $m(P)$ that will modify the algorithm output sufficiently enough to warrant a change in decision:

$$D(A(P)) \neq D(A(m(P)))$$

(1)

Definition 3 still does not provide a good quantitative interpretation of manipulability because it neglects two important factors: the manipulation cost that the manipulator incurs and the effect of the manipulation on the decision maker. The manipulation cost is important because some manipulations could be prohibitively expensive for the manipulator and, therefore, practically infeasible. The manipulator will carry out a manipulative action $m \in M$ on a problem instance $P \in \mathcal{P}$ if the manipulation cost does not overweight the manipulation benefits:

$$U_M(D(A(m(P))), P) > U_M(D(A(P)), P)$$

(2)

On the left side of Equation 2 we have the manipulator’s utility from carrying out a manipulative action $m$ on a problem instance $P$. The right side of the equation represents his utility without manipulation. Because different manipulative actions might have different costs, a rational manipulator must always maximize his utility by choosing the most efficient manipulation $m^*$:

$$m^* = \arg\max_{m \in M} U_M(D(A(m(P))), P)$$

The cost of manipulation draws a demarcation line between theoretical and practical manipulability. Although, a manipulation could exist in theory, it could be prohibitively expensive in practice. A manipulation of a problem instance $P \in \mathcal{P}$ is theoretically possible if there exists a manipulative action $m \in M$ that satisfies Equation 1. Not every manipulative action, however, will satisfy Equation 2 that guarantees cost efficiency for the manipulator. Because different manipulators have different skills and knowledge, what is only theoretically manipulable for one manipulator could be practically manipulable for another.

Another important factor neglected by Definition 3 is the effect the manipulation has on the decision maker. Theoretically, it could be the case that a manipulation might not always hurt the decision maker. For example, a manipulation could make the decision maker reach a decision that is as good as the decision he would reach without manipulation. To exclude such unnatural cases, we assume that there is a conflict of interest between the manipulator and the decision maker in which the decision maker suffers a loss from every manipulation:

$$U_D(D(A(m(P))), P) < U_D(D(A(P)), P)$$

(3)

The left side of Equation 3 represents the decision maker’s utility from a manipulated decision, whereas the right side represents his utility without manipulation. According to the definition, the decision maker regrets making decision, $D(A(m(P)))$, based on the
manipulated output of the algorithm. If he knew the actual problem instance \( P \), he would act differently and decide \( D(A(P)) \) instead of deciding \( D(A(m(P))) \).

The next definition is an improvement of Definition 3 that takes care of the manipulation cost and the effect of the manipulation on the decision maker.

**Definition 4** An algorithm \( A \) is non-manipulable if for every problem instance \( P, \ P \in \mathcal{P} \), there does not exist a manipulative action \( m, \ m \in \mathcal{M} \), such that:

\[
U_M(D(A(m(P))), P) > U_M(D(A(P)), P) \quad \text{and} \quad U_D(D(A(m(P))), P) < U_D(D(A(P)), P)
\]

According to Definition 4, the manipulator must not have an action, \( m \), in his repertoire that allows him to manipulate the problem instance \( P \) to his advantage and to the disadvantage of the decision maker.

### 3. DECISION MAKING BASED ON MANIPULATED INPUT

In this section we discuss the problem of how to make a correct decision based on manipulated input. To solve the problem we first “reverse” the manipulation and identify the real problem instance. Then, we replace the manipulated input with the real input and make a decision based on the real problem instance.

Suppose that the observed problem instance, \( P \), has been manipulated. In this case, it does not make much sense to use the algorithm to solve \( P \) because the results would be probably erroneous. A better alternative would be to find the real problem instance and use it as a basis for a decision. This could be done by working backwards and computing the set of all possible problem instances from which \( P \) could be generated by applying manipulative actions:

\[
\text{Range}(P) = \{ P_k | \exists m \in \mathcal{M}, m(P_k) = P \}
\]

Apparently, the real problem instance belongs to \( \text{Range}(P) \). The problem, however, is that the real problem instance could be buried under many other possible candidates in \( \text{Range}(P) \). The larger \( \text{Range}(P) \) is, the more difficult it is to find the real problem instance, and the greater the confusion effect. In the simplest case, \( \text{Range}(P) \) would have only one element, the real problem instance. In most cases, however, the manipulator would try to hide his intentions by choosing \( \text{Range}(P) \) in which all problem instances seem equally likely. This creates multiple possibilities and complicates the analysis.

Finding \( \text{Range}(P) \) could be computationally expensive. In general, an exhaustive search for \( \text{Range}(P) \) has a running time of \( |\mathcal{M}||\mathcal{P}| \), where \( |\cdot| \) denotes set cardinality. A better approach would be to “reverse” every manipulative action and apply the “reversed” actions on \( P \) to obtain \( \text{Range}(P) \). This method has a running time of \( |\mathcal{M}| \). In other words, for every manipulative action, \( m \in \mathcal{M} \), one needs to compute, \( m^{-1} \), such that if \( m(P_1) = P_2 \),
then \( P_1 \in m^{-1}(P_2) \). In general, \( m^{-1}(P_2) \) could produce more than one problem instance, for example \( m^{-1}(P_2) = \{ P_1, \ldots, P_k \} \).

The next task after finding \( \text{Range}(P) \) is to narrow down the set of possible problem instances by assuming that the manipulator chooses the most effective manipulation, i.e., the manipulation that maximizes his utility. In other words, if the real problem instance was \( P^* \), \( P^* \in \text{Range}(P) \), then the manipulator would choose to manipulate \( P^* \) and produce \( P \) if and only if there is no other manipulation that gives him higher utility:

\[
P = \arg\max_{P^* \in P} U_D(D(A(P^*)), P^*)
\]  

(4)

To find the real problem instance from the observed problem instance \( P \) the decision maker needs to solve Equation (4) for \( P^* \). Then, he needs to feed the real problem instance \( P^* \) into the algorithm and make a decision.

To illustrate the method, consider the following example of a manipulation game in which the role of the manipulator is played by an attacker who wants to reach a target. The decision maker is a defender who must decide how to block the attack. The situation could be conveniently represented by the attack graph shown in Figure 2. The attack graph and the structure of the game, defined in Definition 1, are common knowledge to both players. The notion of attack graph captures the intuition about all possible attacks that can be launched against a set of targets.

Definition 5. An attack graph is a tuple \( AG = < S, A, T, s_0, T> \) where:
- \( S \) is a finite set of states,
- \( A(s) \) is the finite sets of actions available to the attacker in state \( s \in S \),
- \( T \in S \times A \times S \) is the transition relation,
- \( s_0 \in S \) is the initial state,
- \( T \in S \) is the set of possible targets.

![Figure 2: Attack graph](image-url)
Intuitively, the attacker takes actions that move the system from one state to another according to the transition relation $T$. The attacker starts in the initial state, $s_0$, and attempts to reach one of the target states. By definition, the attack stops when it reaches a target state, i.e., $A(s) = \emptyset$ for every target state $s$, $s \in T$. Although Definition 5 is similar to the definition of attack graphs used in computer security (Jha et al. 2002, Phillips and Swiler 1998, Swiler et al. 2001) our model can be applied to a wide range of applications, including military operations, surveillance and intelligence gathering, economic applications, etc. For example, the attacker could be trying to penetrate a market that the defender currently dominates.

The attacker starts at the initial position $S$, chooses an attack route from the attack graph, and tries to reach the target $T$. Each arc is labeled with the cost of traversing the arc. The defender's objective is to protect the target without knowing which particular attack route has been chosen. The situation is complicated by the fact that the defender cannot block all possible routes because of time and resource limitations. In this case, he can block either arc BT or arc FT. Blocking both arcs simultaneously is beyond his means. In addition, blocking arcs takes time and the defender must decide which arc to block right after the attacker has made his first move. We assume that the attacker does not know which arc is blocked and, therefore, cannot switch to another route to bypass the blocked arc. In other words, every blocking action is hidden to the attacker and he needs to decide upfront which attack route to follow.

The game proceeds as follows. The attacker chooses an attack route and follows it until the game ends. The defender observes the first move of the attacker, and based on this observation decides which arc to block: BT or FT. The game ends when the attacker reaches either the target or a blocked arc.

Blocking an arc is equivalent to permanently removing an arc from the attack graph. For example, by blocking arc BT, the attacker simultaneously disables attack routes SABT, SCBT, and SDBT. This, however, is not sufficient to prevent an attack because the attacker could have chosen the attack route SEFT. Similarly, blocking arc FT simultaneously disables attack routes SCFT, SDFT, and SEFT, while keeping routes SABT, SCBT, and SDBT open for an attack.

By observing the attacker's first action the defender needs to predict the route that has been chosen by the attacker. We assume that the defender uses a plan recognition algorithm to predict the attack route. The algorithm takes the attacker’s first move and the attack graph as inputs and produces a prediction of an attack route.

This scenario fits well with our model of algorithm manipulation. Here, the problem instance is a pair, $(first\ move,\ attack\ graph)$, the first element of which is generated by the attacker. The problem instance is fed into the plan recognition algorithm that produces a prediction of an attack route. Based on the prediction, the defender decides to block either BT or FT. Because of the conflict of interest between the attacker and the defender, the attacker might try to manipulate the plan recognition algorithm to his advantage. For example, suppose that the plan recognition algorithm is based on the assumption that the
attacker follows the shortest path to the target. Suppose further, that the attacker has chosen the shortest attack route SABT. Once he reaches point A, the plan recognition algorithm will unambiguously conclude that he has chosen route SABT, because SABT is the only route that passes through point A. To prevent this, the attacker could first visit point D and then, shift back to the original route by visiting B and T. Such manipulation will hide the attacker’s intention and fool the plan recognition algorithm into believing that the attack route is ADFT, because DFT is the shortest path that passes through D. Based on this erroneous prediction, the decision maker will decide to block arc FT and, therefore, lose the game. In this case, visiting point D hides the attacker’s intention and confuses the plan recognition algorithm. The problem is that there are two routes that pass through point D, SDBT and SDFT, and either route could have been chosen by the attacker.

This example suggests that we cannot further assume that the manipulator chooses the most effective manipulation (described by Equation 4) because it will directly reveal the manipulator’s intention. In addition, efficiency might not be the only goal of the manipulator. In many cases, he would hide and obfuscate his intentions in order to increase the probability of success. Therefore, the manipulator might be willing to spend some additional resources for manipulating the algorithm and influencing the decision of the decision maker. Suppose, for example, that the attacker has decided to go through point D and attack the target through arc BT. Although this is the most expensive route he can take, it is worth the extra cost because of the confusion and ambiguity it creates. The attacker could confuse the defender even more by rolling a die at point D and deciding to take route DBT with probability $p$ and route DFT with probability $1-p$. Without loss of generality, we can assume that the attacker rolls a die at the starting point S and randomly takes one attack route. This corresponds to a randomized route:


where each route is taken with probability $p_i$, $i = 1,...,6$. It is obvious that $p_1=p_6=0$ because routes SABT and SEFT will be unambiguously recognized by the plan recognition algorithm after the attacker’s first move. For example, if the attacker’s first move is SA, then $\text{Range}(SA)=\{SABT\}$, there is no manipulation, and the intention of the attacker is clear. On the other hand, if the attacker’s first move is SC, then $\text{Range}(SC)=\{SCBT,SCFT\}$ and both SCBT and SCFT are possible. Apparently, the more elements $\text{Range}$ has, the more difficult it is to find the attacker’s intention.

Because the attacker will not take SABT or SEFT, the randomized route can be simplified to:

$$[SCBT:p_1, SCFT: p_2, SDBT:p_3, SDFT:p_4]$$

where $p_1+p_2+p_3+p_4=0$. Because the attack graph is symmetric, we can search for a symmetric solution in which $p_1=p_4$, and $p_2=p_3$:

$$[SCBT:p/2, SCFT: (1-p)/2, SDBT:(1-p)/2, SDFT:p/2] \quad (5)$$

The best defense against such a random route is a randomized blocking strategy in which the defender blocks arc BT with probability $q$ and arc FT with probability $1-q$. The probability $q$ depends on the attacker’s first move. For instance, if the defender observes
SC as a first move, then he will expect that the attacker will take route CBT with probability $p$ and route CFT with probability $1-p$. In this case, the defender must block arc BT with probability $p$ and arc FT with probability $1-p$ (assuming that blocking BT and blocking FT are equally costly). Similarly, if the defender observes SD as a first move, then he will expect the attacker to take route DFT with probability $p$ and route DBT with probability $1-p$. This observation brings us to the following proposition.

**Proposition.** If blocking BT is as costly as blocking FT, and if the value of the target for the attacker is $G$, then the following strategies are in Nash Equilibrium (Nash 1950):

- **The defender’s strategy:** The defender observes the first move of the attacker. If it is SA, then he blocks BT with probability 1. If it is SC, then he blocks BT with probability $(G+1)/2G$ and FT with probability $(G-1)/2G$. If the first move is SD, the defender blocks BT with probability $(G-1)/2G$ and FT with probability $(G+1)/2G$. Finally, if the attacker’s first move is SE, the defender blocks FT with probability 1.

- **The attacker’s strategy:** The attacker chooses the random route from Equation (5), where $p=(G+1)/2$.

**Proof.** To make the attacker willing to randomize between SCBT, SCFT, SDBT, and SDFT these routes must have the same utility. The utility of SCBT and SDFT is $G(1-p)-4$ and the utility of SCFT and SDBT is $Gp-5$. Therefore,

$$G(1-p) - 4 = Gp - 5$$

The proposition shows that instead of choosing the cheapest manipulation, a rational attacker will randomize between several manipulations in order to hide his intentions. This increases the confusion effect and makes the manipulation difficult to understand.

In general, decision making based on manipulated input involves the following steps:

1. If the manipulated problem instance is P, find $Range(P)$ that includes all possible unmanipulated problem instances. For example, $Range(SC) = \{SCBT, SCFT\}$.

2. Narrow down $Range(P)$ by eliminating those manipulations that do not hide enough the manipulator’s intentions. For instance, routes SABT and SEFT will never be taken by the attacker.

3. Find a randomized manipulation strategy by assigning a probability to each problem instance in $Range(P)$:

   $$Randomized\ manipulation = \{P_1: p_1, P_2:p_2, ..., P_n:p_n\},$$

   where $Range(P) = \{P_1, P_2, ..., P_n\}$.

   For example:

   $$Randomized\ manipulation = [SCBT:p, SCFT: 1-p]$$

4. Find a randomized decision:

   $$Randomized\ decision = [D_1: q_1, D_2:q_2, ..., D_n:q_n]$$

   where $D_i$ is the decision that would be made if the real problem instance was $P_i$. In other words, the decision maker makes a randomized decision in which every
\( D_i \) is made with probability \( q_i \). For example, upon observing that the attacker has moved to point C, the defender decides to block arc BT with probability \( q \), and arc FT with probability \( 1-q \).

5. Solve for \( p_i \) and \( q_i \) so that the randomized manipulation and the randomized decision are in Nash equilibrium. In our example, \( p = q = (G+1)/2G \).

It is very important that the manipulation and the decision strategy are in Nash equilibrium. The equilibrium guarantees that the manipulation is the best response to the decision strategy and vice versa, the decision strategy is the best response to the manipulation. In other words, even if the manipulator knows the internals of the algorithm and the decision function (from Definition 1), he cannot do anything better than to stick to the manipulation. Therefore, the algorithm cannot be further manipulated and it is safe to expose its internals to the public. This satisfies the Kerckhoffs' principle, while preserving the integrity of the decision making process.

CONCLUSIONS

The paper introduces the problem of algorithm manipulation in general and provides a definition that views algorithm manipulation as a game of conflict between a manipulator and a decision maker. The definition takes into account several important factors of algorithm manipulation, such as the possibility and the cost of manipulation, the manipulation incentives, and the effect of the manipulation. It is shown that a rational manipulator should avoid cost-efficient manipulations in order to better hide his intentions.

The paper also presents a general model of decision making based on manipulated data. The method “reverses” possible manipulations and tries to restore the input data to its original state. In order to discourage manipulations, the method proposes a randomized decision making strategy.

Because the paper provides a general treatment of the problem of algorithm manipulation, the framework and the definitions could be applied to a wide range of problems, including adversarial data mining, adversarial learning, adversarial planning and plan recognition, antiterrorist research, and information warfare.

REFERENCES


