ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed
Computing Group

Diploma/Master Thesis

Game theory: (Byzantine) Potential in Games

Raphael Eidenbenz
eraphael@ethz.ch

Prof. Dr. Roger Wattenhofer
Distributed Computing Group

Advisors: Stefan Schmid and Yvonne Anne Oswald

Dept. of Computer Science
Swiss Federal Institute of Technology (ETH) Zurich
March 2007

Mechanism Design by Creditability

Raphael Eidenbenz, Yvonne Anne Oswald, Stefan Schmid, Roger Wattenhofer
{eraphael@, oswald@tik.ee., schmiste@tik.ee., wattenhofer@tik.ee.}ethz.ch
Computer Engineering and Networks Laboratory (TIK), ETH Zurich, 8092 Zurich, Switzerland

Abstract— **This paper attends to the problem of a mechanism designer seeking to influence the outcome of a strategic game based on her creditability. The mechanism designer offers additional payments to the players depending on their mutual choice of strategies in order to steer them to certain decisions. If the players trust her, they will select a newly profitable strategy. Of course, the mechanism designer aims at spending as little as possible and yet implement her desired outcome. We present several algorithms for this optimization problem both for optimal solutions and approximations thereof. We analyze singleton target strategy profiles and target strategy profile sets. Furthermore, the paper shows how a bankrupt mechanism designer can decide efficiently whether strategy profiles can be implemented at no cost at all. Finally risk-averse players and dynamic games are examined.**

I. Introduction

The quest for a deeper understanding of our world and its highly interconnected systems and processes often requires a huge amount of computational resources which can only be obtained by connecting thousands of computers. Similarly to agents in socio-economic systems, the computers in such networks often operate on a decentralized control regime, and represent different stake-holders with different objectives. Therefore, in addition to mere technical challenges, a system designer often has to take into account sociological and economic aspects as well when reasoning about protocols for maximizing system performance.

Game theory is a powerful tool for analyzing decision making in systems with autonomous and rational (or selfish) participants. It is used in a wide variety of fields such as biology, economics, politics, or computer science. A major achievement of game theory is the insight that networks of self-interested agents often suffer from inefficiency due to effects of selfishness. The concept of the *price of anarchy* allows to quantify these effects: The price of anarchy compares the performance of a distributed system consisting of selfish participants to the performance of an optimal reference system where all participants collaborate perfectly. If a game theoretic analysis of a distributed computing system reveals that the system has a large price of anarchy, this indicates that the protocol should be extended by a mechanism encouraging cooperation. The field of *mechanism design* is also subject to active research.

In many distributed systems, a mechanism designer cannot change the rules of interactions. However, she may be able to influence the agents' behavior by offering payments for certain outcomes. On this account, we consider a mechanism designer whose power is to some extent based on her monetary assets, primarily, though, on her *creditability*. That is, the players trust her to pay the promised payments. Thus, a certain subset of outcomes is *implemented* in a given game if, by expecting additional non-negative payments, rational players will necessarily choose one of the desired outcomes. A designer faces the following optimization problem: How can the desired outcome be implemented at minimal cost? Surprisingly, it is sometimes possible to improve the performance of a given system merely by creditability, that is, without any payments at all.

This paper presents several results for this problem. The first correct algorithm for finding an exact, incentive compatible implementation of a desired set of outcomes is given. We also show how a bankrupt mechanism designer can decide in polynomial time if a set of outcomes can be implemented at no costs at all, and an interesting connection to best response graphs is established. We propose and analyze efficient approximation algorithms and demonstrate their performance. Additionally, we extend our analysis for risk-averse behavior and study dynamic games where the mechanism designer offers payments each round.

The remainder of this paper is organized as follows. Section II reviews related work. Our model and some basic game theory definitions are introduced in Section III. In Section IV, algorithms for computing exact and non-exact implementations are proposed. Section V presents simulation results. Risk-averse players and dynamic games are studied in Section VI. Finally, Section VII concludes the paper.

II. Related Work

The mathematical tools of game theory have become popular in computer science recently as they allow to gain deeper insights into the socio-economic complexity of today's distributed systems. Game theory combines algorithmic ideas with concepts and techniques from mathematical economics. Popular problems in computer science studied from a game theoretic point of view include *congestion*

[3], *network creation* [7], or *virus propagation* [1], among many others.

The observation that systems often perform poorly in the presence of selfish players has sparked research for countermeasures [6], [8]. For example, Cole et al. [4], [5] have studied how incentive mechanisms can influence selfish behavior in a routing system where the latency experienced by the network traffic on an edge of the network is a function of the edge congestion, and where the network users are assumed to selfishly route traffic on minimum-latency networks. They show that by pricing network edges the inefficiency of selfish routing can always be eradicated, even for heterogeneous traffic in single-commodity networks and propose algorithms solving these problems.

In [9], Monderer and Tennenholtz consider an interested third party who attempts to lead selfish players to act in a desired way. The third party can neither enforce behavior nor change the system, she can only influence the game's outcome by announcing non-negative monetary transfers conditioned on the behavior of the agents. They show that the interested third party might be able to induce a desired outcome at very low costs. In particular, they prove that any pure Nash equilibrium of a game with complete information has a *zero-implementation*, i.e., it can be transformed into a dominant strategy profile at zero cost. Similar results hold for any given ex-post equilibrium of a frugal VCG mechanism. Moreover, the authors address the question of the hardness of computing the minimal cost. They provide a proof that deciding whether there exists an implementation of cost k for a game is $\mathbf{NP}$-hard and propose an algorithm to compute an optimal exact implementation.

This paper extends [9] in various respects. Several new algorithms are provided, for instance a polynomial time algorithm for deciding whether a set of strategy profiles has a 0-implementation. In addition, we suggest polynomial-time approximation algorithms and simulate their performance. Connections to graph-theoretic concepts are pointed out and we generalize the theorem by Monderer and Tennenholtz theorem on the cost of Nash equilibria. Their algorithm for computing an optimal exact implementation is corrected, and we provide evidence that their $\mathbf{NP}$-hardness proof of deciding whether a k-implementation exists is wrong. Furthermore, the concept of implementation is generalized for other game theoretic models. We introduce players aiming at maximizing the average payoff and show how the mechanism designer can find such implementations. As another contribution, this paper considers the case of risk-averse players as well as the resulting complexity of computing the optimal implementation cost, and initiates the study of mechanism design by creditability in round based dynamic games.

Our work is also related to *Stackelberg theory* [10] where a fraction of the entire population is orchestrated by a global leader. In contrast to our paper, the leader is not bound to offer any incentives to follow her objectives. Finally, in the recent research thread of *combinatorial agencies* [2], a setting is studied where a mechanism designer seeks to influence the outcome of a game by contracting the players individually; however, as she is not able to observe the players' actions, the contracts can only depend on the overall outcome.

III. Model

This section first reviews some basic definitions and formalisms from game theory. Subsequently, the concept of mechanism design by creditability is introduced.

A. Game Theory

A *strategic game* can be described by a tuple $G = (N, X, U)$. $N = \{1, 2, \ldots, n\}$ is the set of *players* and each player $i \in N$ can choose a *strategy* (action) from the set X_i. The product of all the individual players' strategies is denoted by $X := X_1 \times X_2 \times \ldots \times X_n$. In the following, a particular outcome $x \in X$ is called *strategy profile* and we refer to the set of all other players' strategies of a given player i by $X_{-i} = X_1 \times \ldots \times X_{i-1} \times X_{i+1} \times \ldots \times X_n$. An element of X_i is denoted by x_i, and similarly, $x_{-i} \in X_{-i}$; hence x_{-i} is a vector consisting of the strategy profiles of x_i. Finally, $U = (U_1, U_2, \ldots, U_n)$ is an n-tuple of *payoff functions*, where $U_i : X \mapsto \Re$ determines player i's payoff arising from the game's outcome.

Let $x_i, x_i' \in X_i$ be two strategies available to player i. We say that x_i *dominates* x_i' iff $U_i(x_i, x_{-i}) \geq U_i(x_i', x_{-i})$ for every $x_{-i} \in X_{-i}$ and there exists at least one x_{-i} for which a strict inequality holds. x_i is the *dominant* strategy for player i if it dominates every other strategy $x_i' \in X_i \backslash \{x_i\}$. x_i is a *non-dominated* strategy if no other strategy dominates it. By $X^* = X_1^* \times X_2^* \times \ldots \times X_n^*$ we will denote the set of non-dominated strategy profiles, where X_i^* is the set of non-dominated strategies available to the individual player i. We assume that players are rational and always choose non-dominated strategies; furthermore players cannot collude or make contracts to choose certain profiles beforehand.

The set of *best responses* $B_i(x_{-i})$ for player i given the other players' actions is defined as $B_i(x_{-i}) := \{\arg\max_{x_i \in X_i}(U_i(x_i, x_{-i}))\}$. A *Nash equilibrium* is a strategy profile $x \in X$ such that for all $i \in N$, $x_i \in B_i(x_{-i})$.

B. Mechanism Design by Creditability

This paper makes the classic assumption that players are rational and always choose a non-dominated strategy. We examine the impact of payments to players offered by a *mechanism designer* (an interested third party) who seeks to influence the outcome of a game. These payments are described by a tuple of non-negative payoff functions $V = (V_1, V_2, \ldots, V_n)$, where $V_i : X \mapsto \Re$, i.e. the payments depend on the strategies player i selects as well as on the choices of all other players. The original game $G = (N, X, U)$ is modified to $G(V) := (N, X, [U + V])$ by these payments, where $[U + V]_i(x) = U_i(x) + V_i(x)$, that is, each player i obtains the payoff of V_i in addition to the payoffs of U_i. The players' choice of strategies changes accordingly: Each player now selects a non-dominated strategy in $G(V)$. Henceforth, the set of non-dominated strategy profiles of $G(V)$ is denoted by $X^*(V)$. A *strategy profile set* – also called *strategy profile region* – $O \subseteq X$ of G is a subset of all strategy profiles X, i.e., a region in the payoff matrix consisting of one or multiple strategy profiles. Similarly to X_i and X_{-i}, we define $O_i := \{x_i | \exists x_{-i} \in X_{-i} \text{ s.t. } (x_i, x_{-i}) \in O\}$ and $O_{-i} := \{x_{-i} | \exists x_i \in X_i \text{ s.t. } (x_i, x_{-i}) \in O\}$.

The mechanism designer's main objective is to force the players to choose a certain strategy profile or a set of strategy profiles, without having to spend too much. Concretely, for a desired strategy profile region O, we say that payments V *implement* O if $\emptyset \subset X^*(V) \subseteq O$. V is called *k-implementation* – a mechanism design by credibility with cost at most k – if, in addition $\sum_{i=1}^{n} V_i(x) \leq k \ \forall x \in X^*(V)$. That is, the players' non-dominated strategies are within the desired strategy profile, and the payments do not exceed k for any possible outcome. Moreover, V is an *exact k-implementation* of O if $X^*(V) = O$ and $\sum_{i=1}^{n} V_i(x) \leq k \ \forall x \in X^*(V)$. The *costs* $k(O)$ of implementing O is the lowest of all non-negative numbers q for which there exists a q-implementation. If an implementation meets this lower bound, it is optimal, i.e., V is an *optimal implementation* of O if V implements O and $\max_{x \in X^*(V)} \sum_{i=1}^{n} V_i(x) = k(O)$. The cost $k^*(O)$ of implementing O exactly is the smallest non-negative numbers q for which there exists an exact q-implementation of O. V is an *optimal exact implementation* of O if it implements O exactly and requires the costs $k^*(O)$. The set of all implementations of O will be denoted by $\mathcal{V}(O)$, and the set of all exact implementations of O by $\mathcal{V}^*(O)$. Finally, a strategy profile region $O = \{z\}$ of cardinality one – consisting of only one strategy profile – is called a *singleton*. Clearly, for singletons it holds that non-exact and exact k-implementations are equivalent. For simplicity's sake we often write z instead of $\{z\}$ and $V(z)$ instead of $\sum_{i \in N} V_i(z)$. Observe, that only subsets of X which are in $2^{X_1} \times 2^{X_2} \times \ldots \times 2^{X_n} \subset 2^{X_1 \times X_2 \times \ldots \times X_n}$ can be implemented exactly. We call such a subset of X a *convex strategy profile region*.[1]

IV. Algorithms and Analysis

This section presents our main results. First, we study exact implementations where the mechanism designer aims at implementing an *entire* strategy profile region based on her creditability. Subsequently, we examine general k-implementations.

A. Exact Implementation

Recall that the matrix V is an exact k-implementation of a strategy region O iff $X^*(V) = O$ and $\sum_{i=1}^{n} V_i(x) \leq k \ \forall x \in X^*(V)$, i.e. each strategy O_i is part of the set of player i's non-dominated strategies for all players i. We present the first correct algorithm to find such implementations. Then we show that a bankrupt mechanism designer can determine in polynomial time whether a given region is implementable at zero cost. We will also establish an interesting connection between zero cost implementations and best response graphs.

1) Algorithm and Complexity: Recall that in our model each player classifies the strategies available to her as either dominated or non-dominated. Thereby, each dominated strategy $x_i \in X_i \backslash X_i^*$ is dominated by at least one strategy $x_i^* \in X_i^*$ among the non-dominated strategies. In other words, a game determines for each player i a relation M_i^G from dominated to non-dominated strategies $M_i^G : X_i \backslash X_i^* \rightarrow X_i^*$, where $M_i^G(x_i) = x_i^*$ states that $x_i \in X_i \backslash X_i^*$ is dominated by $x_i^* \in X_i^*$. See Fig. 1 for an example.

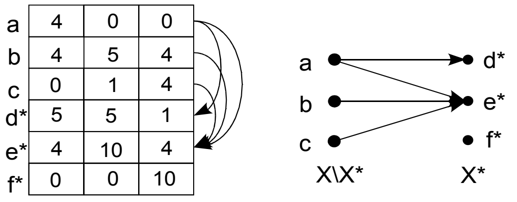


Fig. 1. Game from a single player's point of view with corresponding relation of dominated ($X_i \backslash X_i^* = \{a, b, c\}$) to non-dominated strategies ($X_i^* = \{d^*, e^*, f^*\}$).

When implementing a strategy profile region O exactly, the mechanism designer creates a modified game $G(V)$ with a new relation $M_i^V : X_i \setminus O_i \rightarrow O_i$ such that all strategies outside O_i map to at least one strategy in O_i. Therewith, the set of all newly non-dominated strategies of player i must constitute O_i. As every $V \in \mathcal{V}^*(O)$ determines a set of relations $M^V := \{M_i^V : i \in N\}$, there must be a set M^V for every V implementing O optimally as well. If we are given such an optimal relation set M^V, but not the corresponding optimal exact implementation, we can compute a V with minimal payments and the same relation M^V, i.e., given an optimal relation we can find an optimal exact implementation. As an illustrating example, assume an optimal relation set for G with $M_i^G(x_{i1}^*) = o_i$ and $M_i^G(x_{i2}^*) = o_i$. Thus, we can compute V such that o_i must dominate x_{i1}^* and x_{i2}^* in $G(V)$. To attain this, the condition $U_i(o_i, o_{-i}) + V_i(o_i, o_{-i}) \geq \max_{s \in \{x_{i1}^*, x_{i2}^*\}} (U_i(s, o_{-i}) + V_i(s, o_{-i}))$ must hold for all $o_{-i} \in O_{-i}$. In an optimal implementation, player i is

[1]These regions define a convex area in the payoff matrix, provided that the strategies are depicted such that all o_i are next to each other.

not offered payments for strategy profiles of the form $(\bar{o}_i, x_{-i})$ where $\bar{o}_i \in X_i \backslash O_i$, $x_{-i} \in X_{-i}$. Hence, the condition above can be simplified to $V_i(o_i, o_{-i}) = \max\left\{0, \max_{s \in \{x^*_{i1}, x^*_{i2}\}}\left(U_i(s, o_{-i})\right)\right\} - U_i(o_i, o_{-i})$. Let $S_i(o_i)$:=$\{s \in X_i \backslash O_i | M_i^V(s) = o_i\}$ be the set of strategies where M^V corresponds to an optimal exact implementation of O. Then, an implementation V with $V_i(\bar{o}_i, x_{-i}) = 0$, $V_i(o_i, \bar{o}_{-i}) = \infty$ for any player i, and $V_i(o_i, o_{-i}) = \max\left\{0, \max_{s \in S_i(o_i)}\left(U_i(s, o_{-i})\right)\right\} - U_i(o_i, o_{-i})$ is an optimal exact implementation of O as well. Therefore, the problem of finding an optimal exact implementation V of O corresponds to the problem of finding an optimal set of relations $M_i^V : X_i \backslash O_i \rightarrow O_i$.

Our algorithm $\mathcal{ALG}_{exact}$ (cf Algorithm 1) makes use of this fact and constructs an implementation V for all possible relation sets, checks the cost that V would produce and returns the lowest cost found.

Algorithm 1 Exact k-Implementation ($\mathcal{ALG}_{exact}$)

```
Input: Game $G$, convex region $O$ with $O_{-i} \subset X_{-i} \forall\ i$
Output: $k^*(O)$
 1: $V_i(x) := 0$, $W_i(x) := 0\ \forall x \in X\ ,\ i \in N$;
 2: $V_i(o_i, \bar{o}_{-i}) := \infty\ \forall i \in N$, $o_i \in O_i\ ,\ \bar{o}_{-i} \in X_{-i} \backslash O_{-i}$;
 3: return ExactK($V$, 1);
ExactK($V$, $i$):
Input: payments $V$, current player $i$
Output: minimal $r$ s.t. $\exists$ exact $r$-implementation
    $W \in \{W | W(x) \geq V(x)\ \forall x \in X\}$
 1: if $|X_i^*(V) \backslash O_i| > 0$ then
 2:     $s :=$ any strategy in $X_i^*(V) \backslash O_i$; $k_{best} := \infty$;
 3:     for all $o_i \in O_i$ do
 4:         for all $o_{-i} \in O_{-i}$ do
 5:             $W(o_i, o_{-i})$:=max$\{0, U_i(s, o_{-i}) -$
                $(U_i(o_i, o_{-i}) + V(o_i, o_{-i}))\}$;
 6:         $k :=$ ExactK($V + W$, $i$);
 7:         if $k < k_{best}$ then
 8:             $k_{best} := k$;
 9:         for all $o_{-i} \in O_{-i}$ do
10:             $W(o_i, o_{-i}) := 0$;
11:     return $k_{best}$;
12: else if $i < n$ then
13:     return $ExactK(V, i + 1)$;
14: else
15:     return $\max_{o \in O} \sum_i V_i(o)$;
```

Theorem 4.1: $\mathcal{ALG}_{exact}$ computes a strategy profile region's optimal exact implementation cost in time $O\left(n\,|X| + n\left(\max_{i \in N} |O_i|^{n \max_{i \in N} |X_i^*|}\right)\right)$.

PROOF. $\mathcal{ALG}_{exact}$ is correct as it checks all possible relations in relation set $M^V = \{M_i^V : X_i^*(V) \backslash O_i \mapsto O_i\ \forall i \in N\}$ recursively by calling the subroutine $ExactK$ in Line 6. Therefore, it must find the relation set which corresponds to an implementation with optimal cost.

In order to analyze $\mathcal{ALG}_{exact}$'s runtime, we first examine the complexity of subroutine *ExactK*. Observe that *ExactK* is called $|O_i|$ times per recursion step. In each recursion step there is at least one non-dominated strategy, namely $s \in X_i^* \backslash O_i$, which is dominated in the next call to *ExactK*. As there are $s = \sum_{i=1}^n |X_i^* \backslash O_i| \leq n \max_{i \in N} |X_i^* \backslash O_i| \leq n \max_{i \in N} |X_i^*|$ strategies to be dominated, there are at most s recursion steps. Hence, we get the following recursive formula for the runtime of *ExactK* $T(s) = |O_i| \left[|O_{-i}| + T(s-1) + 1 + |O_{-i}|\right] = |O_i| \left(1 + 2\,|O_{-i}|\right) + |O_i| \left[T(s-1)\right]$ where $T(0) = |O|\,n$. Telescoping yields $T(s) = \sum_{k=1}^s \left(1 + 2\,|O_{-i}|\right) |O_i|^k$ $+T(0)\,|O_i|^s$=$\sum_{k=1}^s \left(1 + 2\,|O_{-i}|\right) |O_i|^k$ $+n\,|O|\,|O_i|^s \in \mathrm{O}(n\,|O|\,|O_i|^s)$. The initialization phase (Lines 1 & 2) takes time $T_{init} = \mathrm{O}(n\,|X| + \sum_{i=1}^n |O_i|\,|O_{-i}|) = \mathrm{O}(n\,|X| + n\,|O|)$. Therefore, the overall time complexity $T_{init} + T(s)$ is in $\mathrm{O}\left(n\,|X| + n\left(\max_{i \in N} |O_i|^{n \max_{i \in N} |X_i^*|}\right)\right)$. □

Note that $\mathcal{ALG}_{exact}$ has a large time complexity. In fact, a faster algorithm for this problem, called *Optimal Perturbation Algorithm* has been presented in [9]. In a nutshell, this algorithm proceeds as follows: After initializing V similarly to our algorithm, the values of the region O in the matrix V are increased slowly for every player, i.e., by all possible differences between an agent's payoffs in the original game. The algorithm terminates as soon as all strategies in X^* are dominated. Unfortunately, this algorithm does not always return an optimal implementation. Sometimes, as we will show in Appendix A, the optimal perturbation algorithm increases the values too much. In fact, we even conjecture that deciding whether an k-exact implementation exists is **NP**-hard.

Conjecture 4.1: Finding an optimal exact implementation of a strategy region is **NP**-hard in general.

2) Bankrupt Mechanism Designers: Imagine a mechanism designer who is broke. At first sight, it seems that without any money, she will hardly be able to influence the outcome of a game. However, this intuition ignores the power of creditability: a game can have 0-implementable regions.

Let V be an exact implementation of O with exact costs $k^*(O)$. It holds that if $k^*(O) = 0$, V cannot contain any payments larger than 0 in O. Consequently, for an region O to be 0-implementable exactly, any strategy s outside O_i must be dominated within the range of O_{-i} by a o_i, or there must be one o_i for which no payoff $U_i(s, o_{-i})$ is larger than $U_i(o_i, o_{-i})$. In the latter case, the strategy o_i can still can dominate s by using a payment $V(o_i, x_{-i})$ with $x_{-i} \in X_{-i} \backslash O_{-i}$ outside O. Note that this is only possible under the assumption that $O_{-i} \subset X_{-i} \forall i \in N$.

$\mathcal{ALG}_{bankrupt}$ (cf Algorithm 2) describes how a bankrupt designer can decide in polynomial time whether a certain region is 0-implementable. It proceeds by checking for each player i if the strategies in $X_i^* \backslash O_i$ are dominated or "almost" dominated within the range of O_{-i} by at least one strategy inside O_i. If there is one strategy without such a dominating strategy, O is not 0-implementable exactly. On the other hand, if for every strategy $s \in X_i^*\backslash O_i$ such a dominating strategy is found, O can be implemented exactly without expenses.

Algorithm 2 Exact 0-Implementation ($\mathcal{ALG}_{bankrupt}$)

```
Input: Game $G$, convex region $O$ with $O_{-i} \subset X_{-i} \forall i$
Output: true if $k^*(O) = 0$, false otherwise
 1: for all $i \in N$ do
 2:     for all $s \in X_i^* \backslash O_i$ do
 3:         dZero := false;
 4:         for all $o_i \in O_i$ do
 5:             b := true;
 6:             for all $o_{-i} \in O_{-i}$ do
 7:                 b := b $\wedge$ ($U_i(s, o_{-i}) \leq U_i(o_i, o_{-i})$);
 8:             dZero := dZero $\vee$ b;
 9:         if !dZero then
10:             return false;
11: return true;
```

Theorem 4.2: Given a convex strategy profile region O with $O_{-i} \subset X_{-i} \forall i$, $\mathcal{ALG}_{bankrupt}$ decides whether O has an exact 0-implementation in time $\mathrm{O}(|X^*|\,|O|)$.

PROOF. $\mathcal{ALG}_{bankrupt}$ is correct because it checks for each yet to be dominated strategy $s \in X_i^* \backslash O_i$ whether it can be dominated by one $o_i \in O_i$ at zero cost. This is the property that makes O exactly 0-implementable. The algorithm's runtime is maximal if X^* does not intersect with O, and if O is indeed 0-implementable, summing up to $\mathrm{O}\big(\sum_{i \in N} |X_i^*| \cdot |O_i| \cdot |O_{-i}|\big) = \mathrm{O}\big(\sum_{i \in N} |X_i^*| \cdot |O|\big) = \mathrm{O}(|X^*| \cdot |O|)$. □

3) Best Response Graphs: Best response strategies maximize the payoff for a player given the other players' decisions are x_{-i}. For now, let us restrict our analysis to games where the sets of best response strategies consist of only one strategy for each $x_{-i} \forall i \in N$. Given a game G, we can construct a directed *best response graph* $\mathcal{G}_G$ as follows: A strategy profile $x \in X$ is represented as a vertex v_x in $\mathcal{G}_G$ iff x is a best response for at least one player, i.e., if $\exists i \in N$ such that $x_i \in B_i(x_{-i})$. There is a directed edge $e = (v_x, v_y)$ iff $\exists i \in N$ such that $x_{-i} = y_{-i}$ and $\{y_i\} = B_i(y_{-i})$. In other words, if there is an edge from v_x to v_y, this means that for one player, it is better to play y_i than x_i given that for the other players' strategies $x_{-i} = y_{-i}$. A strategy profile region $O \subset X$ has a *corresponding subgraph* $\mathcal{G}_{G,O}$ containing the vertices $\{v_x | x \in O\}$ and the edges which both start and end in a vertex of the subgraph. We say $\mathcal{G}_{G,O}$ has an *outgoing edge* $e = (v_x, v_y)$ if $x \in O$ and $y \notin O$. Note that outgoing edges are not in the edge set of $\mathcal{G}_{G,O}$. Clearly, it holds that if a singleton x's corresponding subgraph $\mathcal{G}_{G,\{x\}}$ has no outgoing edges then x is a *Nash equilibrium*. More generally, we make the following observation.

Theorem 4.3: Let G be a game and $|B_i(x_{-i})| = 1 \; \forall i \in N, x_{-i} \in X_{-i}$. If a convex region O has an exact 0-implementation, then the corresponding subgraph $\mathcal{G}_{G,O}$ in the game's best response graph has no outgoing edges.

PROOF. Let V be an exact 0-implementation of O. Note that $V(o) = 0 \; \forall o \in O$, otherwise the cost induced by V are larger than 0. Assume for the sake of contradiction that $\mathcal{G}_{G,O}$ has an outgoing edge. Let $x \in O$ be a strategy profile for which its corresponding vertex v_x has an outgoing edge e to $v_y, y \in X \backslash O$. Since $V(x)$ is 0, $\mathcal{G}_{G(V),O}$ still has the same outgoing edge e. This means that for one player j it is better to play strategy y_j in $G(V)$ than to play x_j given that $x_{-j} = y_{-j}$. Hence, since y_j is not dominated by any strategy in O_j, player j will choose also strategies outside O_j and therefore V is not a correct implementation of O thus contradicting our assumption. □

In order to expand best response graphs to games with multiple best responses, we construct $\mathcal{G}_G$ such that Theorem 4.3 still holds. We modify the edge construction as follows: In the general best response graph $\mathcal{G}_G$ of a game G there is a directed edge $e = (v_x, v_y)$ iff $\exists i \in N$ s.t. $x_{-i} = y_{-i}$, $y_i \in B_i(y_{-i})$ and $|B_i(y_{-i})| = 1$.

Corollary 4.1: Theorem 4.3 holds for general games.

Note that Theorem 4.3 is a generalization of Monderer and Tennenholtz' Corollary 1 in [9]. They discovered that for a singleton x, it holds that x has a 0-implementation if and only if x is a Nash equilibrium. While their observation covers the special case of singleton-regions, our theorem holds for any strategy profile region. Unfortunately, for general regions, one direction of the equivalence holding for singletons does not hold anymore due to the fact that 0-implementable regions O must contain a player's best response to any o_{-i} but they need not contain best responses exclusively.

5 4	5 5	5 1
10 10	10 0	0 10

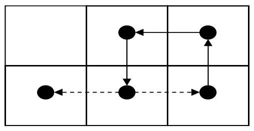


Fig. 2. Sample game G with best response graph $\mathcal{G}_G$. The Nash equilibrium in the bottom left corner has no outgoing edges. The dotted arrows do not belong to the edge set of $\mathcal{G}_G$ as the row has multiple best responses.

B. Non-Exact Implementation

In contrast to exact implementations, where the complete set of strategy profiles O must be non-dominated, the additional payments in non-exact implementations only have to ensure that a *subset* of O is the newly non-dominated region. Obviously, it matters which subset this is. Knowing that a subset $o' \subseteq O$ bears optimal costs one, we could find $k(O)$ by computing $k^*(O')$. Thus, when solving the problem of finding an optimal implementation V of a region O, the problem of finding among all subsets of O a subset with minimal exact cost is solved as well. Apart from the fact that finding an optimal implementation includes solving the – believed to be $\mathbf{NP}$-hard – optimal exact implementation cost problem for at least one subregion of O, finding this subregion might be $\mathbf{NP}$-hard since there are exponentially many possible subregions in O. In fact, a proof is given in [9] showing that the problem is $\mathbf{NP}$-hard by a reduction from the SAT problem. The authors show how to construct a 2-person game in polynomial time given a CNF formula such that the game has a 2-implementation if and only if the formula has a satisfying assignment. However, their proof is not correct: While there indeed exists a 2-implementation for every satisfiable formula, it can be shown that 2-implementations also exist for non-satisfiable formulas. E.g., strategy profiles $(x_i, x_i) \in O$ are always 1-implementable. Unfortunately, we were not able to correct their proof. However, we are strongly believe that the problem is $\mathbf{NP}$-hard and we assume that no algorithm can do much better than performing a brute force computation of the exact implementation costs (cf Algorithm 1) of all possible subsets, unless $\mathbf{NP} = \mathbf{P}$.

Conjecture 4.2: Finding an optimal implementation of a strategy region is $\mathbf{NP}$-hard in general.

Although the optimization problem of finding a strategy profile region's cost seems to be hard, the special case of zero cost regions allows a statement related to best response graphs. From the fact that if V is a 0-implementation, V is also an exact 0-implementation of $O' = X^*(V)$, Theorem 4.3 implies Corollary 4.2.

Corollary 4.2: If a strategy profile region O has zero implementation cost then the corresponding subgraph $\mathcal{G}_{G,O}$ in the game's best response graph contains a subgraph $\mathcal{G}_{G,O'}, O' \subseteq O$, with no outgoing edges.
This could be useful to a bankrupt mechanism designer since searching the game's best response graph for subgraphs without outgoing edges helps her spot candidates for regions which can be implemented by mere creditability.

In general though, the fact that finding optimal implementations seems computationally hard raises the question whether there are polynomial time algorithms achieving good approximations. As mentioned in Section IV-A, each V implementing a region O defines a domination relation $M_i^V : X_i \setminus O_i \to O_i$. This observation leads to the idea of designing approximation algorithms that find a correct implementation by establishing a corresponding relation set $\{M_1, M_2, \ldots, M_n\}, M_i : X_i^* \backslash O_i \mapsto O_i$ where each $x_i^* \in X_i^* \backslash O_i$ maps to at least one $o_i \in O_i$. These algorithms are guaranteed to find a correct implementation of O, however, these implementations are not guaranteed to be cost-optimal.[2]

[2]Admitting approximation algorithms to slightly deviate from the desired region might open another class of algorithms. However, such considerations are beyond the scope of this paper.

Our greedy algorithm $\mathcal{ALG}_{greedy}$ (cf Algorithm 3) associates each strategy x_i^* yet to be dominated with the o_i with minimal distance Δ_G to x_i^*, i.e., the maximum value that have to be added to $U_i(x_i', x_{-i})$ such that x_i' dominates x_i: $\Delta_G(x_i, x_i') := \max_{x_{-i} \in X_{-i}} \max \{0, U_i(x_i, x_{-i}) - U_i(x_i', x_{-i})\}$.

Similarly to the greedy approximation algorithm for the *set cover problem* [**?**], [**?**] which chooses in each step the subset covering the most elements not covered already, $\mathcal{ALG}_{greedy}$ selects a pair of x_i^*,o_i such that by dominating x_i^* with o_i, the number of strategies in $X_i^* \backslash O_i$ that will be dominated therewith is maximal. Thus, in each step there will be an o_i assigned to dominate x_i^* which has minimal dominating cost. Additionally, if there is the chance of dominating multiple strategies, $\mathcal{ALG}_{greedy}$ takes it.

$\mathcal{ALG}_{greedy}$ is described in detail in Algorithm 3. Note that it returns an implementation V of O. To determine V's cost, one can compute $\max_{x^* \in X^*(V)} \sum_{i \in N} V_i(x^*)$.

Algorithm 3 Greedy Algorithm $\mathcal{ALG}_{greedy}$

```
Input: Game $G$, convex target region $O$
Output: Implementation $V$ of $O$
 1: $V_i(x) := 0; W_i(x) := 0 \ \forall x \in X \ , \ i \in N$;
 2: for all $i \in N$ do
 3:     $V_i(o_i, \bar{o}_{-i}) := \infty \ \forall o_i \in O_i \ , \ \bar{o}_{-i} \in X_{-i} \backslash O_{-i}$;
 4:     while $X_i^*(V) \not\subseteq O_i$ do
 5:         $c_{best} := 0; m_{best} :=$null; $s_{best} :=$null;
 6:         for all $s \in X_i^*(V) \backslash O_i$ do
 7:             $m := \arg\min_{o_i \in O_i} \left[\Delta_{G(V)}(s, o_i)\right]$;
 8:             for all $o_{-i} \in O_{-i}$ do
 9:                 $W(m, o_{-i})$:=max$\{0, U_i(s, o_{-i}) -$
                    $(U_i(m, o_{-i}) + V(m, o_{-i}))\}$;
10:             $c := 0$;
11:             for all $x \in X_i^*$ do
12:                 if $m$ dominates $x$ in $G(V + W)$ then
13:                     $c++$;
14:             if $c > c_{best}$ then
15:                 $c_{best} := c$ ; $m_{best} := m$ ; $s_{best} := s$;
16:         for all $o_{-i} \in O_{-i}$ do
17:             $V(m_{best}, o_{-i})$+=max$\{0, U_i(s_{best}, o_{-i}) -$
                $(U_i(m_{best}, o_{-i}) + V(m_{best}, o_{-i}))\}$;
18: return $V$;
```

Theorem 4.4: $\mathcal{ALG}_{greedy}$ returns an implementation of O in $\mathrm{O}\Big(n\,|X| + n\,|O| \max_{i \in N} \left[|X_i^* \backslash O_i|^3 \, |O_{-i}|^2\right]\Big)$.
PROOF. $\mathcal{ALG}_{greedy}$ terminates since in every iteration of the while-loop, there is at least one newly dominated strategy. The payment matrix V returned is an implementation of O because the while-condition $X_i^*(V) \not\subseteq O_i$ turned false for all $i \in N$ and thus, it holds that $X_i^*(V) \subseteq O_i \ \forall i \in N$. The algorithm's initialization phase (Line 1) takes time $T_{init} \in \mathrm{O}(|X|\, n)$. Setting the payments $V_i(o_i, \bar{o}_{-i})$ to infinity (Line 3) for all players takes $T_{inf} \in \mathrm{O}(n\,|X \backslash O|)$. One iteration of the while loop (Lines 4-17) takes $T_{while} \in \mathrm{O}([|X_i^*(V) \backslash O_i|\,|O|\,|O_{-i}|\,|X_i^*(V)|\,|O_{-i}| + |O_{-i}|]) =$

Algorithm 4 Reduction Algorithm $\mathcal{ALG}_{red}$

```
Input: Game $G$, convex target region $O$
Output: Implementation $V$ of $O$
 1: $[k, V] := greedy(G, O)$;
 2: $k_{temp} := -1$; $c_i := \bot\ \forall i$; $T_i := \{\}$;
 3: while $(k > 0) \wedge (\exists i : |O_i| > 1) \wedge (\exists i : O_i \nsubseteq T_i)$ do
 4:     for all $i \in N$ do
 5:         $x_i := \arg\min_{o_i \in O_i} \left[\max_{o_{-i} \in O_{-i}} U_i(o_i, o_{-i})\right]$;
 6:         if $(O_i \nsubseteq T_i) \wedge \neg(\forall j: |T_j| = 0 \vee c_j) \wedge (x_i \in T_i)$ then
 7:             $x_i := \arg\min_{o_i \in O_i \setminus \{x_i\}} \left[\max_{o_{-i} \in O_{-i}} U_i(o_i, o_{-i})\right]$;
 8:         if $|O_i| > 1$ then
 9:             $O_i := O_i \setminus \{x_i\}$;
10:         $[k_{temp}, V] := greedy(G, O)$;
11:         if $k_{temp} \geq k$ then
12:             $O_i := O_i \cup \{x_i\}$; $T_i := T_i \cup \{x_i\}$; $c_i := \bot$;
13:         else
14:             $k := k_{temp}$; $T_i := \{\}\ \forall i$; $c_i := \top$;
15: return $V$;
```

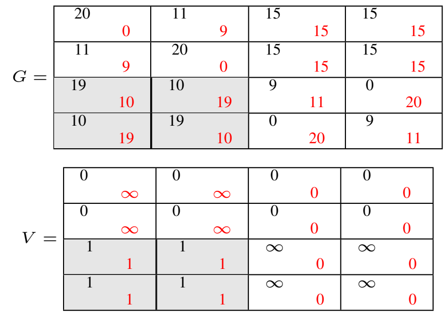
$O\left(|O|\,|X_i^* \backslash O_i|^2\,|O_{-i}|^2\right)$. Consequently, the total time is $O\left(T_{init} + T_{inf} + \sum_{i \in N} \left(|X_i^*(V) \backslash O_i|\, T_{while}\right)\right) \in O\left(n\,|X| + n\,|O| \max_{i \in N}\left[|X_i^* \backslash O_i|^3\,|O_{-i}|^2\right]\right)$. □

$\mathcal{ALG}_{red}$ (cf Algorithm 4) is a more sophisticated version of our greedy algorithm. Instead of terminating when the payment matrix V implements O, this algorithm continues to search for a payment matrix inducing even less cost. It uses a greedy algorithm to approximate the cost repeatedly, varying the region to be implemented. As $\mathcal{ALG}_{greedy}$ leaves the while loop if $X_i^*(V) \subseteq O_i$, it might miss out on cheap implementations where $X_i^*(V) \subseteq Q_i,\ Q_i \subset O_i$. $\mathcal{ALG}_{red}$ examines some of these subsets as well by calling a greedy algorithm for some Q_i, see Lines 5-10. If we manage to reduce the cost (Lines 13-20), we continue with $O_i := Q_i$ until neither the cost can be reduced anymore nor any strategies can be deleted from any O_i. Moreover, $\mathcal{ALG}_{red}$ loops over all $i \in N$ in every execution of the while loops and thus profits even more from the removal of strategies.

Theorem 4.5: $\mathcal{ALG}_{red}$ returns an implementation of O in time $O\left(T_g n \max_{i \in N} |O_i| \sum_{i \in N} |O_i|\right)$, where T_g denotes the runtime of a greedy algorithm.

PROOF. $\mathcal{ALG}_{red}$ terminates because the condition of the while loop does not hold anymore if there are no more strategies to be removed. By using $\mathcal{ALG}_{greedy}$ the correctness of the algorithm follows from Theorem 4.4. Let T_g denote the runtime of the greedy algorithm called in Line 1 and 10. A worst case instance requires the algorithm to remove all but one strategy profiles, at least one in every $\max_{i \in N} |O_i|^{th}$ iteration of the while loop. As looping over all i in Line 4 takes $n \cdot T_g$ and there are $\sum_{i \in N} |O_i|$ distinct strategy profiles the total runtime is in $O\left(nT_g \max_{i \in N} |O_i| \sum_{i \in N} |O_i|\right)$ □

A much simpler approximation algorithm for computing a region O's implementation cost retrieves the region's cheapest singleton, i.e., $\min_{o \in O} k(o)$. As proven in [9], there is a simple formula for a singleton's implementation cost, namely $k(o) = \min_{o \in O} \sum_{i \in N} \max_{x_i \in X_i} \left(U_i(x_i, o_{-i}) - U_i(o_i, o_{-i})\right)$. Thus, implementing a best singleton approximation algorithm is straight forward and it performs quite well with randomly generated games – as our simulations show (cf Section V). However, this approach can result in an arbitrarily large k for certain games. Fig. 3 depicts such a game where each singleton o in the region O consisting of the four bottom left profiles has cost $k(o) = 11$ whereas V implements O with cost 2.



$G =$

20, 0	11, 9	15, 15	15, 15
11, 9	20, 0	15, 15	15, 15
19, 10	10, 19	9, 11	0, 20
10, 19	19, 10	0, 20	9, 11

$V =$

0, ∞	0, ∞	0, 0	0, 0
0, ∞	0, ∞	0, 0	0, 0
1, 1	1, 1	∞, 0	∞, 0
1, 1	1, 1	∞, 0	∞, 0

Fig. 3. 2-player game where O's optimal implementation V yields a non-singleton region $|X^*(V)| > 1$.

This raises the following question: What characteristics are stringent for a game and a corresponding desired strategy profile region O such that only non-singleton subregions bear the optimal implementation cost? Clearly, we have to consider games where at least one player has four or more strategies, at least two of which must not be in O_i. Moreover, it must cost less to dominate them with two strategies in O_i than with just one strategy in O_i.

V. SIMULATION

All our algorithms return correct implementations of the desired strategy profile sets and – apart from the recursive algorithm $\mathcal{ALG}_{exact}$ for the optimal exact implementation – run in polynomial time. In order to study the quality of the resulting implementations, we performed several simulations comparing the implementation costs computed by the different algorithms. We have focused on *bimatrix games*, i.e., two-person games, using random game tables where for each strategy profile, both players have a random payoff chosen uniformly at random from the interval $[0, max]$, for some constant max. We have also studied generalized *scissors, rock, paper games* (a.k.a., *Jan Ken Pon games*), that is, *symmetric zero-sum games* with payoff values chosen uniformly at random from an interval $[0, max]$. We find that – for the same interval and the same number of strategies – the average implementation cost of random symmetric zero-sum games, random symmetric games, and completely random games hardly deviate. This is probably due to the fact that in all examined types of random games virtually all strategies are non-dominated. Therefore, in the following, we present our results on symmetric random games only.

Our simulations give several interesting insights. Let us attend to non-exact implementations first. We observe that implementing the best singleton often yields low costs. In other words, especially when large sets have to be implemented, our greedy algorithms tend to implement too many strategy profiles and consequently incur unnecessarily high costs. However, while this is often true in random games, there are counter examples where the cheapest singleton is costly compared to the implementation found by the greedy algorithms; Fig. 3 depicts a situation where the greedy algorithm computes a better solution. We presume that the $\mathcal{ALG}_{red}$ might improve relatively to the best singleton approximation algorithm for larger player sets.

Fig. 4 plots the implementation costs determined by the greedy algorithm $\mathcal{ALG}_{greedy}$, the reduction algorithm $\mathcal{ALG}_{red}$, and the singleton algorithm as a function of the number of strategies involved. On average, the singleton algorithm performed much better than the other two, with $\mathcal{ALG}_{greedy}$ being the worst of the candidates. In the second plot we can see the implementation costs the algorithms compute for different payoff value intervals $[0, max]$. As expected, the implementation cost increases for larger intervals.

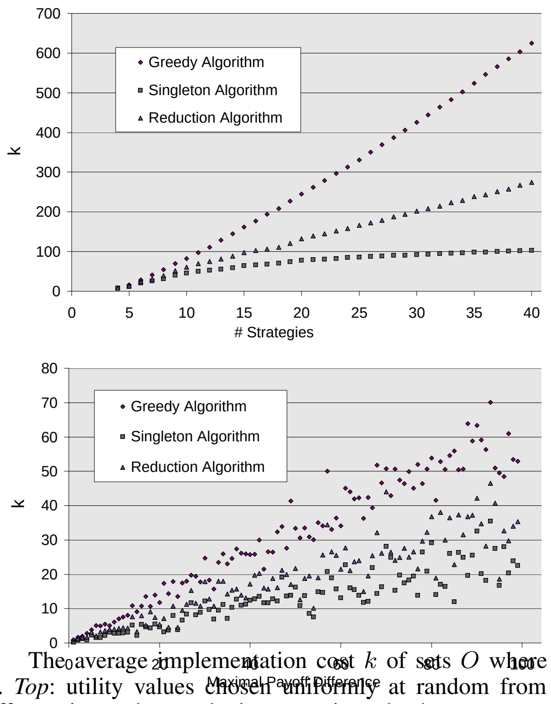


Fig. 4. The average implementation cost k of sets O where $|O_i| = \lfloor n/3 \rfloor$. *Top*: utility values chosen uniformly at random from $[0, 20]$. For different intervals we obtain approximately the same result when normalizing k with the maximal possible value. *Bottom*: eight strategies are used; other numbers of strategies bear similar results.

The observation that the greedy algorithm $\mathcal{ALG}_{greedy}$ implements rather large subregions of O suggests that it may achieve good results for exact implementations. Indeed, we can modify an implementation V of O, which yields a subset of O, without modifying any entry $V_i(o), o \in O$, such that the resulting V implements O *exactly*.

Theorem 5.1: Let V implement O. If $\forall i \in N$, $O_{-i} \subset X_{-i}$, it holds that $k^*(O) \leq \max_{o \in O} V(o)$.

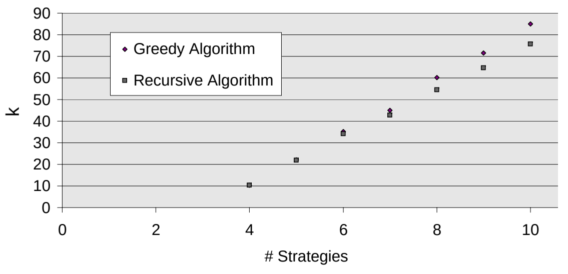
PROOF. If V is a non-exact implementation of O, there are some strategies O_i dominated by other strategies in O_i. A dominated strategy o_i can be made non-dominated by adding payments to the existing V_i for profiles of the form $(o_i, \bar{o}_{-i})$, where $\bar{o}_{-i} \in X_{-i} \backslash O_{-i}$. Let $a \in O_i$ dominate $b \in O_i$ in $G(V)$. The interested party can annihilate this relation of a dominating b by choosing payment $V_i(b, \bar{o}_{-i})$ such that player i's resulting payoff $U_i(b, \bar{o}_{-i}) + V_i(b, \bar{o}_{-i})$ is larger than $U_i(a, \bar{o}_{-i}) + V_i(a, \bar{o}_{-i})$ and therefore a does not dominate b anymore. As such, all dominations inside O_i can be neutralized even if $|O_{-i} \subset X_{-i}| = 1$. To see this one must realize that the relation of domination is *irreflexive* and *transitive* and therefore establishes a strict order among the strategies. Let $\bar{o}_{-i} \in X_{-i} \backslash O_{-i}$ be one column in player i's payoff matrix outside O. By choosing the payments $V_i(o_i, \bar{o}_{-i})$ such that the resulting payoffs $U_i(o_i, \bar{o}_{-i}) + V_i(o_i, \bar{o}_{-i})$ establish the same order with the *less-than relation* ($<$) as the strategies o_i with the domination relation, all $o_i \in O_i$ will be non-dominated. Thus, a V' can be constructed from V which implements O exactly without modifying any entry $V_i(o), o \in O$. $\square$

Theorem 5.1 enables us to use $\mathcal{ALG}_{greedy}$ for an exact cost approximation by simply computing $\max_{o \in O} V(o)$ instead of $\max_{x \in X^*(V)} V(x)$.

Fig. 5 depicts the exact implementation costs determined by the greedy algorithm $\mathcal{ALG}_{greedy}$ and the optimal algorithm $\mathcal{ALG}_{exact}$. The first figure plots k as a function of the number of strategies, whereas the second figure demonstrates the effects of varying the size of the payoff interval. Due to the large runtime of $\mathcal{ALG}_{exact}$, we were only able to compute k for a small number of strategies. However, for these cases, our simulations reveals that $\mathcal{ALG}_{greedy}$ often finds implementations which are close to optimal. For different payoff value intervals $[0, max]$, we observe a faster increase in k than in the non-exact implementation case. This suggests that implementing a smaller region entails lower costs for random games on average .

Finally, we tested different options to choose the next strategy in Line 7 of $\mathcal{ALG}_{greedy}$ and in Line 8 of $\mathcal{ALG}_{red}$. However, none of the alternatives we tested performed better than the ones described in Section IV.

In conclusion, our simulations have shown that for the case of non-exact implementations, there are interesting differences between the algorithms proposed in Section IV. In particular, the additional reductions made be $\mathcal{ALG}_{red}$ are beneficial. However, while it seems that



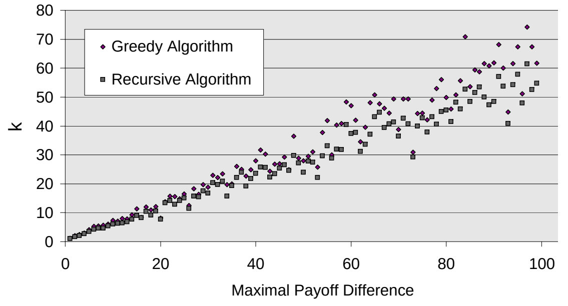


Fig. 5. The average exact implementation cost k of sets O where $|O_i| = \lfloor n/3 \rfloor$. *Top*: utility values chosen uniformly at random from $[0, 20]$. For other intervals we obtain approximately the same result when normalizing k with the maximal possible value. *Bottom*: eight strategies are used; again, the plot is similar for other numbers of strategies.

finding good approximations is hard, there may still be ways to improve on the existing approximations. For the case of exact implementations, our modified greedy algorithm yields good results.

VI. Variations

Mechanism design by creditability offers many interesting extensions. In this section, two alternative models of rationality are introduced. If we assume that players do not just select *any* non-dominated strategy, but have other parameters influencing their decision process, our model has to be adjusted. In many (real world) games, players typically do not know which strategies the other players will choose. In this case, a player cannot do better than assume that the other players select a strategy at random. If a player wants to maximize her game under this assumption, she can take the *average payoff* of strategies into account. We study the consequences of this process of decision making. In a second part of this section, risk-averse players are examined. Finally, we take a brief look at the dynamics of repeated games with an interested third party offering payments in each round.

A. Average Payoff Model

As a player may choose any non-dominated strategy, it is reasonable for a player i to compute the payoff which each of her strategy will bring *on average*. Thus, assuming no knowledge on the payoffs of the other players, each strategy x_i has an average payoff of $p_i(x_i) := \frac{1}{|X_{-i}|} \sum_{x_{-i} \in X_{-i}} U_i(x_i, x_{-i})$ for player i. The player i will then choose the strategy $s \in X_i$ with the largest $p_i(s)$, i.e., $s = \arg\max_{s \in X_i} p_i(s)$. If multiple strategies have the same average payoff, she plays one of them uniformly at random. For such average strategy games, we say that x_i *dominates* x'_i iff $p_i(x_i) > p_i(x'_i)$. Note that with this modified meaning of domination, the region of non-dominated strategies, X^*, differs as well.

The average payoff model has interesting properties. It can be shown that singleton profiles can always be implemented for free.

Theorem 6.1: If players maximize their average payoff, singleton strategy profiles are always 0-implementable if there are at least two players with at least two strategies.

PROOF. Let z be the strategy profile to be implemented. In order to make player i choose strategy z_i, the interested party may offer payments for any strategy profile $(z_i, \bar{z}_{-i})$ where $\bar{z}_{-i} \in X_{-i} \setminus \{z_{-i}\}$ such that $p_i(z_i)$ becomes player i's largest average payoff in $G(V)$. Since each player has at least two strategies to choose from, there is at least one $\bar{z}_{-i}$, and by making $V_i(z_i, \bar{z}_{-i})$ large enough (e.g., $V_i(z_i, \bar{z}_{-i}) := \max_{x_i \in X_i} \sum_{x_{-i} \in X_{-i}} U_i(x_i, x_{-i}) + \epsilon$) this can always be achieved. Therefore, z can be implemented without promising any payments for z. □

Clearly, Theorem 6.1 implies that also entire strategy profile regions O are 0-implementable: We can simply implement any singleton inside O.

Corollary 6.1: In average strategy games where every player has at least two strategies, each strategy profile region can be implemented for free.

Not only non-exact implementations, but also exact implementations can be implemented at no costs.

Theorem 6.2: In average strategy games where $\forall i$: $O_{-i} \subset X_{-i}$, each strategy profile region has an exact 0-implementation.

PROOF. The mechanism designer can proceed as follows. For player i, let $\mu_i := \max_{x_i \in X_i}\{p_i(x_i)\}$. We set $V_i(o_i, \bar{o}_{-i}) := |X_{-i}|(\mu_i - p_i(x_i)) - U_i(x_i, x_{-i}) + \epsilon, \forall o_i \in O_i$, $\bar{o}_{-i} \in X_{-i} \setminus O_{-i}$. Consequently, it holds that for each player i and two strategies $x_i \in O_i$ and $x'_i \notin O_i$, $p_i(x_i) > p_i(x'_i)$; moreover, no strategy $x_i \in O_i$ is dominated by any other strategy. As payments in $V_i(o_i, \bar{o}_{-i})$ with $o_i \in O_i$ and $\bar{o}_{-i} \in X_{-i} \setminus O_{-i}$ do not contribute to the implementation cost, Theorem 6.2 follows. □

B. Risk-Averse Players

Instead of striving for a high payoff on average, the players might be cautious or *risk-averse*. To account for such behavior, we adapt our model by assuming that the players seek to minimize the risk on missing out on benefits. In order to achieve this objective, they select strategies where the minimum gain is not less than any other strategy's minimum gain. If there is more than one strategy with this property, the risk-averse player can choose a strategy among these, where the average of the benefits is maximal. More formally, let $min_i := \max_{x_i \in X_i}(\min_{x_{-i} \in X_{-i}}(U_i(x_i, x_{-i})))$ and $\varnothing_X f(x) := \frac{1}{|X|} \cdot \sum_{x \in X} f(x)$. Then P_i selects a strategy m satisfying $m = \arg\max_{m \in M}(\varnothing_{X_{-i}} U_i(m, x_{-i}))$, where $M = \{x_i | \forall x_{-i} \quad U_i(x_i, x_{-i}) = min_i\}$.

Theorem 6.3: The implementation cost of a singleton $z \in X$ is $k = \sum_{i=1}^{N} \max(0, min_i - U_i(z))$ for risk-averse players.

PROOF. We show how to construct V implementing z with cost k and then prove that we cannot reduce the payments of $V(z)$. Since in this model every player i makes her decision without taking into account the benefits other players might or might not obtain, it suffices to consider each V_i separately. To ensure that player i selects z_i we have to set $V_i(z_i, x_{-i})$ to a value such that min_i is reached in $G(U+V)$ for each x_{-i}. Consequently we assign $V_i(z_i, x_{-i}) = \max(0, min_i - U_i(z_i, x_{-i}))$. We have to satisfy a second condition such that z_i is chosen, namely, $z_i = \arg\max_{m \in M}(\varnothing_{X_{-i}} U_i(m, x_{-i}))$. This is achieved by setting $V_i(z_i, x_{-i}) = \infty \quad \forall x_{-i} \neq z_{-i}$. We repeat these steps for all players i. Clearly V constructed in this manner implements z. Since the cost k only comprises the additional payments in $V(z)$ and lowering $V_i(z)$ for any i results in player i choosing a different strategy, we can deduce the statement of the theorem. □

For desired regions, the situation with risk-averse players differs from the standard model considerably.

Theorem 6.4: For risk-averse players the cost of $O \subset X$ is $k(O)$=$\min_{o \in O} \sum_{i=1}^{N} \max(0, min_i - U_i(o))$.

PROOF. Since we have to add up the cost to reach the required minimum for every strategy profile in $X^*(V)$ it cannot cost less to exactly implement more than one strategy profile, i.e., find V such that $|X^*(V)| = 1$. Thus V implementing the "cheapest" singleton in O provides an optimal implementation for O, and the claim follows. □

In Section IV, we conjecture the problem of computing $k(O)$ to be $\mathbf{NP}$-complete for both general and exact implementations. Risk-averse players change the situation, as the following theorem states.

Theorem 6.5: The complexity of computing k for risk-averse agents is in $\mathbf{P}$.

PROOF. For the non-exact case this theorem follows directly from Theorem 6.4. In order to prove Theorem 6.5 for exact implementations we demonstrate how to compute $V_i(o)$ such that V implements the entire region O optimally. For a player i and a set of strategies $Y_i \subseteq X_i$, we define $\tau(Y_i)$:=$\max_{x_i \in Y_i}(\varnothing_{X_{-i}}((U+V)_i(x_i, x_{-i})))$ to be the maximum of the average benefits over all strategies. For each strategy of a player i, we define $\delta(x_i)$:=$\max(\tau(O_i), \tau(X_i^*)) - \varnothing_{X_{-i}}((U+V)_i(x_i, x_{-i})))$, for $x_i \in X_i$, to be the difference of the averages. Algorithm 5 constructs V if the target region O and X^* are disjoint. Analogously to the proofs above we can deal with each player i individually. It computes for all cases how much the interested party has to offer at least for strategy profiles in O and sets $V_i(x_i, x_{-i})$ to infinity for all $x_i \in O_i$, $x_{-i} \in X_{-i} \setminus O_{-i}$ (Line 3). Then, for each player i, strategies O_i have to reach the minimum payoff of strategies in X_i^* (Line 4). This suffices for an exact implementation if $O_i \subset X_i$, i.e. if there exists at least one strategy $x_i \notin O_i$. Otherwise, we determine whether it costs more to exceed the minimum constraint or the average constraint for all O_i if O_i covers whole columns (Lines 7 and 8) and adjust V_i accordingly (average in Lines 8 and 13, minimum in Lines 10 and 15). Thus the algorithm ensures that only strategies in O are chosen while all strategies in O are selected.

Algorithm 5 Risk-averse Players: Exact Implementation

```
Input: Game $G$, target region $O$, $O_i \cap X_i^* = \emptyset \ \forall i \in N$
Output: $V$
1:  $V_i(z) = 0$ for all $i \in N, z \in X$;
2:  for all $i \in N$ do
3:      $V_i(x_i, x_{-i})$:=$\infty \ \forall x_i \in O_i,\ x_{-i} \in X_{-i} \setminus O_{-i}$;
4:      $V_i(x_i, x_{-i}) := \max(0, min_i - U_i(x_i, x_{-i})) \ \forall x_i \in O_i,$
        $x_{-i} \in X_{-i}$;
5:      if $O_{-i} = X_{-i}$ then
6:          if $\tau(O_i) > \tau(X_i^*)$ then
7:              if $|X_i| + \epsilon|O_i| > |X_i| + \sum_{o_i} \delta(o_i)$ then
8:                  $V_i(o_i, x_{-i})$:=$V_i(o_i, x_{-i}) + \delta(o_i) \ \forall o_i, x_{-i}$;
9:              else
10:                 $V_i(o_i, x_{-i})$:=$V_i(o_i, x_{-i}) + \epsilon \ \forall o_i, x_{-i}$;
11:         else
12:             if $\epsilon|O_i| > \sum_{o_i} [\epsilon + \delta(o_i)]$ then
13:                 $V_i(o_i, x_{-i})$:=$V_i(o_i, x_{-i}) + \epsilon + \delta(o_i) \forall o_i, x_{-i}$;
14:             else
15:                 $V_i(o_i, x_{-i})$:=$V_i(o_i, x_{-i}) + \epsilon \ \forall o_i, x_{-i}$;
16: return $V$;
```

The algorithm can be extended easily to work for instances where $X_i^* \subset O_i$. As the extension is straightforward and does not provide any new insights, we omit it. The run time of the algorithm can be determined to be $O(N \max_{i \in N}(|O_i||X_{-i}|))$, thus we can compute $k^*(O) = \max_{o \in O} V(o)$ in polynomial time. □

C. Round-based Mechanisms

The previous sections deal with a static model. Now, we extend our analysis to dynamic, round-based games, where the designer offers payments to the players after each round in order to make them change strategies. This opens many questions: For example, imagine a concrete game such as a *network creation game* [7] where all players are stuck in a costly Nash equilibrium. The goal of a mechanism designer could then be to guide the players into another Nash equilibrium with lower costs. Many such extensions are reasonable; due to space constraints, we present only one model.

In a dynamic game, we regard a strategy profile as a state in which the participants find themselves. In a network context, each $x \in X$ could represent one particular network topology. We presume to find the game in an initial starting state $s^{T=0} \in X$ and that, in state $s^{T=t}$, each player i only sees the states she can reach by changing her strategy given the other players remain with their chosen strategies. Thus player i sees only strategy profiles in $X^{T=t}_{visible,i} = X_i \times \{s^{T=t}_{-i}\}$ in round t. In every round t, the mechanism designer offers the players a payment matrix $V^{T=t}$ (in addition to the game's static payoff matrix U). Then all players switch to their best visible strategy (which is any best response $B_i(s^{T=t}_{-i})$), and the game's state changes to $s^{T=t+1}$. Before the next round starts, the mechanism designer disburses the payments $V^{T=t}(s^{T=t+1})$ offered for the newly reached state. The same procedure is repeated until the mechanism designer decides to stop the game.

We prove that a mechanism designer can guide the players to any strategy profile at zero costs in two rounds.

Theorem 6.6: Starting in an arbitrary strategy profile, a dynamic mechanism can be designed to lead the players to any strategy profile without any expenses in at most two rounds if $|X_i| \geq 3\ \forall i \in N$.

In order to simplify the proof of Theorem 6.6, we begin with a helper lemma. Let $X^{T=t}_{visible}$ denote the *visible strategy profile region* in round t, that is, $X^{T=t}_{visible} = \bigcup_{i=1}^{n} X^{T=t}_{visible,i}$.

Lemma 6.1: The third party can lead the players of a dynamic game to any strategy profile outside the visible strategy profile region without any expenses in one round.

PROOF. Let $s \in X$ be the starting strategy profile and e the desired end strategy profile in the non-visible region of s. The designer can implement e in just one round by offering each player i an infinite amount $V_i(x)$ for the strategy profile $x = (e_i, s_{-i})$ and zero for any other. Thus each player will switch to e_i. Since $V_i((e_i, s_{-i}))$ are the only positive payments offered and since all $x = (e_i, s_{-i})$ are visible and e is non-visible from s which implies $e \neq x$, hence e is implemented at no cost. □

PROOF OF THEOREM 6.6. Consider an arbitrary starting strategy profile s and a desired strategy profile e. If e is not visible from s, e is implementable at no cost in one round, as seen in Lemma 6.1. If e is visible from s, the interested party can still implement e for free by taking a detour to a strategy profile d which is neither in s' visible region nor in e's visible region. Such a strategy profile d exists if the player i who sees e from s has at least 3 strategies to choose from and $|X_{-i}| \geq 2$. See Fig. 6 for an illustration of such a configuration. □

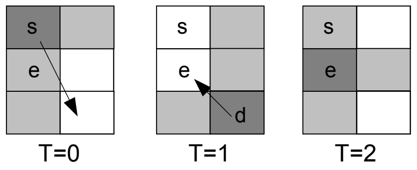


Fig. 6. A dynamic game: Starting in **s**, strategy profile **e** can be implemented at zero cost within two rounds by taking a detour to **d**. The colored region marks the visible strategy profiles at each step.

VII. CONCLUSION

Rendering distributed systems robust to non-cooperative behavior has become an important research topic. This paper has studied the fundamental question: Which outcomes can be implemented by promising players money while the eventual payments are bounded? As a first contribution, we have presented algorithms for various objectives. In addition, efficient algorithms have been suggested yielding implementations of low cost. We have found that for the case of exact implementations, a greedy algorithm performs quite well for random games. Finally, we have introduced the study of round-based games and risk-averse players. We believe that our extensions raise some exciting questions for future research.

REFERENCES

[1] J. Aspnes, K. Chang, and A. Yampolskiy. Inoculation Strategies for Victims of Viruses and the Sum-of-Squares Partition Problem. In *Proc. 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 43–52, 2005.

[2] M. Babaioff, M. Feldman, and N. Nisan. Combinatorial Agency. In *Proc. 7th ACM Conference on Electronic Commerce (EC)*, pages 18–28, 2006.

[3] G. Christodoulou and E. Koutsoupias. The Price of Anarchy of Finite Congestion Games. In *Proc. 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 67–73, 2005.

[4] R. Cole, Y. Dodis, and T. Roughgarden. How Much Can Taxes Help Selfish Routing? In *Proc. 4th ACM Conference on Electronic Commerce (EC)*, pages 98–107, 2003.

[5] R. Cole, Y. Dodis, and T. Roughgarden. Pricing Network Edges for Heterogeneous Selfish Users. In *Proc. 35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 521–530, 2003.

[6] R. Dash, D. Parkes, and N. Jennings. Computational Mechanism Design: A Call to Arms. In *IEEE Intelligent Systems*, 2003.

[7] A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and S. Shenker. On a Network Creation Game. In *Proc. 22nd Annual Symposium on Principles of Distributed Computing (PODC)*, pages 347–351, 2003.

[8] D. S. Johnson. Approximation Algorithms for Combinatorial Problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[9] L. Lovász. On the Ratio of Optimal Integral and Fractional Covers. *Discrete Mathematics*, 13:391–398, 1975.

[10] E. Maskin and T. Sjöström. *Handbook of Social Choice Theory and Welfare (Implementation Theory)*, volume 1. North-Holland, Amsterdam, 2002.

[11] D. Monderer and M. Tennenholtz. k-Implementation. In *Proc. 4th ACM Conference on Electronic Commerce (EC)*, pages 19–28, 2003.

[12] T. Roughgarden. Stackelberg Scheduling Strategies. In *Proc. ACM Symposium on Theory of Computing (STOC)*, pages 104–113, 2001.

Appendix

Non-Optimality of the Perturbation Algorithm

In this section, we give an example demonstrating that the optimal perturbation algorithm presented in [9] is not correct. The algorithm computes the payoff matrix V for the following game G, see Fig. 7.

$$G = \begin{array}{|c|c|}\hline 2,\ 0 & 0,\ 0 \\ \hline 0,\ 0 & 2,\ 3 \\ \hline 4,\ 0 & 0,\ 0 \\ \hline \end{array}$$

$$V_1 = \begin{array}{|c|c|}\hline 2,\ 0 & 5,\ 0 \\ \hline 0,\ 3 & 5,\ 0 \\ \hline 0,\ 5 & 0,\ 0 \\ \hline \end{array} \quad V_2 = \begin{array}{|c|c|}\hline 2,\ 3 & 5,\ 0 \\ \hline 2,\ 3 & 5,\ 0 \\ \hline 0,\ 5 & 0,\ 0 \\ \hline \end{array}$$

Fig. 7. Game G with X^* and O , payoff matrices $V1$, $V2$.

As can be verified easily, V_1 implements O with cost $k = 3$. The payoff matrix V_2 computed by the optimal perturbation algorithm implements O as well, however, it has cost $k = 5$. The set of possible differences between an agent's payoffs in the original game for G is $E = \{0, 2, 3, 4\}$. We execute Lines 2 and 3 and obtain a matrix of perturbation G' of

$$G' = \begin{array}{|c|c|}\hline 0,\ 0 & 5,\ 0 \\ \hline 0,\ 0 & 5,\ 0 \\ \hline 0,\ 5 & 0,\ 0 \\ \hline \end{array}$$

We have to go through the Lines 4 to 8 for player 1 twice, for $e_1 = 0$ and $e_2 = 2$, generating $G'(Player_1, e_1), G'(Player_1, e_2)$ respectively, such that afterwards $(G + G')^*_1$ coincides with O_1.

$$G'(Player_1, e_1) = \begin{array}{|c|c|}\hline 0,\ 0 & 5,\ 0 \\ \hline 0,\ 0 & 5,\ 0 \\ \hline 0,\ 5 & 0,\ 0 \\ \hline \end{array}$$

$$G'(Player_1, e_2) = \begin{array}{|c|c|}\hline 2,\ 0 & 5,\ 0 \\ \hline 2,\ 0 & 5,\ 0 \\ \hline 0,\ 5 & 0,\ 0 \\ \hline \end{array}$$

Executing Lines 9 to 13 until the condition in Line 13 is satisfied and hence $(G + G')^*_2 \equiv O_2$ results in

$$G'(Player_2, e_1) = G'(Player_1, e_2)$$

$$G'(Player_2, e_2) = \begin{array}{|c|c|}\hline 2,\ 2 & 5,\ 0 \\ \hline 0,\ 2 & 5,\ 0 \\ \hline 0,\ 5 & 0,\ 0 \\ \hline \end{array}$$

$$G'(Player_1, e_3) = \begin{array}{|c|c|}\hline 2,\ 3 & 5,\ 0 \\ \hline 0,\ 3 & 5,\ 0 \\ \hline 0,\ 5 & 0,\ 0 \\ \hline \end{array}$$

Thus, the perturbation algorithm returns the payoff matrix $V = G'$.

(Byzantine) Potentials in Games

Raphael Eidenbenz, Yvonne Anne Oswald, Stefan Schmid, Roger Wattenhofer
{eraphael@, oswald@tik.ee., schmiste@tik.ee., wattenhofer@tik.ee.}ethz.ch
Computer Engineering and Networks Laboratory (TIK), ETH Zurich, 8092 Zurich, Switzerland

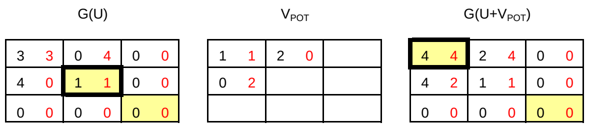
***Abstract*— This paper introduces the notion of potential in games which captures the extent to which the social welfare can be improved or worsened within economic reason, i.e., by taking the implementation cost into account. We prove that there are games where it is worth spending money on influencing the players' decisions. Often a malicious mechanism designer can corrupt games and worsen the players' situation to a by far larger extent than the amount of money she invests. Surprisingly, she sometimes even needs no money at all. We provide an optimal implementation algorithm for this problem, extend the research on implementation mechanisms by creditability with an expected cost analysis, give a NP-hardness proof, and present a greedy algorithm. Finally, two interesting variations of our models are discussed: Average payoff decision-making and round-based implementation mechanisms.**

I. Introduction

Algorithmic game theory and mechanism design has become a popular tool to gain insights into the socio-logical, economical and political complexities of today's distributed systems such as politics, global markets or the Internet. Typically, when a game-theoretic analysis reveals that a system may suffer from selfish behavior, appropriate countermeasures have to be taken in order to enforce a desired behavior.

As it is often infeasible in a distributed system for a mechanism designer to influence the rules according to which the players act, he has to resort to other measurements. One way of manipulating the players' decision-making is to offer them money for a certain behavior. At first sight, such a mechanism does not sound promising since a self-interested agent will simply do whatever brings him the most money. However, Monderer and Tennenholtz [5] have come up with a trick to outwit selfish agents whereby in some cases no money is needed at all to implement a certain behavior. They propose to offer the players money *depending* on how the game turns out. Thus, the question of which monetary offers the designer – also referred to as interested party – has to eventually disburse depends on the other players' decisions. Nevertheless, there is no information hidden and every decision is made on a rational basis.

Consider the following extension of the well-known prisoner's dilemma game (cf $G(U)$ in Figure 1) where two bank robbers, both members of the Capone clan, are arrested by the police. Unfortunately, the policemen have insufficient evidence for convicting them of robbing the bank. However they could charge them a minor crime, since both of them are well-known small-time criminals. Cleverly, the policemen interrogate each subject separately and offer both of them the same deal. If one testifies to the fact that his accomplice has participated in the bank robbery they do not charge him for the minor crime.


G(U)
V_POT
G(U+V_POT)

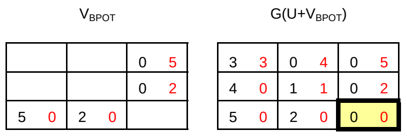
3	3	0	4	0	0
4	0	1	1	0	0
0	0	0	0	0	0

1	1	2	0		
0	2				

4	4	2	4	0	0
4	2	1	1	0	0
0	0	0	0	0	0

Fig. 1. Prisoners' Dilemma. The left bimatrix shows the prisoners' initial payoffs, where payoff values equal saved years. The first strategy is to remain silent (s), the second to testify (t) and the third to confess (c). Nash equilibrias are colored yellow. Non-dominated strategy profiles have a thick border. The center bimatrix V shows Mr. Capone's offered payments which modify G to the game $G(V)$ on the right. Implementing the upper left corner strategy profile (s, s) costs 2 and improves the players welfare by 4. Hence (s, s) has potential=2.

Thus the following scenarios arise. If one testifies and the other remains silent, the former goes free and the latter receives a three-year sentence for robbing the bank and a one-year sentence for committing the minor crime. If both betray the other they both get three years for the bank robbery. If none betrays the other, the police can convict them only for the minor crime, they get one year each. There is another option, of course, to confess to the entire bank robbery and thus supply the police with evidence to convict both criminals for a four-year sentence. In standard game theoretic decision-making, each prisoner has to decide for a strategy, either he remains silent (s) or he testifies against the other (t) or he confesses that they robbed the bank together (c). It shows that it is always the best strategy, no matter what the accomplice does, to choose t. Thus the prisoners would betray each other and both get charged a three-year sentence. This is very much to the dislike of Mr. Capone who would loose two of his best men for the next three years. Luckily enough, he gets a chance to take influence on his employees' decisions since the policemen give the robbers some more decision-taking time and put them on remand. Before they take their decision, Mr. Capone calls each of them and promises that if they both remain silent, he will pay each one of them money worth going to jail one year[1] and moreover, if one remains silent and the other betrays him he will pay the former money worthy of two years in prison (cf V in Figure 1). By offering these payments, Mr. Capone creates a new situation for the two criminals where remaining silent is now the most rational behavior. Indeed, neither one of the bank robbers betrays the other and they both get charged a one-year sentence. One year later, they are released and Mr. Capone disburses them the promised amount of money. Obviously, he is not happy about the fact that his men got arrested, but he prefers to pay them extra money for one year than to do without them for another two years. Thus, he has saved his clan an accumulated two years in jail.



V_BPOT
G(U+V_BPOT)

		0 5
		0 2
5 0	2 0	

3 3	0 4	0 5
4 0	1 1	0 2
5 0	2 0	0 0

Fig. 2. Prisoners' Dilemma. By offering payments V_{BPOT} in game G of Figure 1, the police implements the right bottom corner strategy profile (c, c). As $V_1(c, c) = V_2(c, c) = 0$, payments V implement (c, c) for free. The players' welfare is worsened by 2. Hence, strategy profile (c, c) has a Byzantine Potential of 2.

Let us consider a slightly different scenario where after the police officers proposed their offer to the prisoners, their commander-in-chief comes up with an even more promising deal. He offers each criminal to drop two years of the four-year sentence in case he confesses the entire bank robbery and his accomplice betrays him. Moreover, if he confesses all and the accomplice remains silent they would let him go free and even reward his honesty with a share of the booty (worth going to prison for one year). However, if both suspects confess the entire coup, they will both go to prison for four years. In this new situation it is most rational for a prisoner to confess. Thus, since both are presented the same deal, they will both confess and therefore be charged a four year-sentence. With this trick, the commander-in-chief accomplishes to implement the best outcome in his point of view without dropping any sentence and he increases the accumulated years in prison by two.

From Mr. Capone's point of view, implementing the outcome where both prisoners keep quiet brings the Capone Clan two years. Thus, the strategy profile where both prisoners play s has a potential of two. From the police's point of view, the strategy profile where both prisoners play c has a potential of two. Since implementing c worsens the player's payoff, we say the strategy profile where both play c has *Byzantine* potential of two.

In the described scenario, Mr. Capone and the commander-in-chief solve the optimization problem of finding the game's strategy profile(s) which bear largest (Byzantine) potential and therewith the problem of implementing the corresponding outcome at optimal cost. This paper analyzes these problems' complexity and presents algorithms for finding the potential of games in different models. We show that while some tasks can be solved in polynomial time by the mechanism designer, some tasks are NP-hard. We believe that our notion of potentials raises interesting algorithmic questions in a variety of models beyond standard game theory, some of which are presented briefly in this paper as well.

The remainder of this paper is organized as follows. In the next section (Section II), related literature is reviewed and compared to our work. We introduce some formalism and our model in Section III. Section IV then presents various algorithms for computing the potential of a game. The subsequent section, Section V studies the case of less risk averse players and shows how so-called expected implementation cost is computed. Section VI introduces expected potential analysis and presents algorithms for finding a game's expected potential. Two interesting extensions of our model are presented in Section VII, namely average payoff and dynamic games, before Section VIII concludes the paper.

II. Related Work

Research in the area of algorithmic mechanism design and game theory is very active and it is impossible to give a complete overview. For good introductions to game theory and microeconomic theory see [6] and [4], respectively.

[1]For this scenario, we presume that time really is money!

The notion of game potential is related to the notion of k-implementation tossed by Monderer and Tennenholtz [5]. They consider an interested third party who cannot enforce behaviors and cannot change the system, and who attempts to lead agents to adopt desired behaviors in a given multi-agent setting. The only way the third party can influence the outcome of the game is by promising non-negative monetary transfers conditioned on the observed behavior of the agents. They show that surprisingly, the interested third party might be able to induce a desired outcome at very low costs (cf also [8]). The authors provide algorithms for computing the minimal cost required for implementing a certain outcome of the game. However, these cost are not compared to the corresponding changes of the social cost they entail. We aim at filling this gap by introducing the notion of game potential.

Eidenbenz et al. [2] have continued the analysis of [5] and have provided deeper insights about the possibilities and algorithmic complexities of mechanisms based on creditability. They provide algorithms for computing a strategy profile region's implementation cost and extended the notion of k-implementation to round-based games, risk-averse player games and average payoff games.

Our work is also related to *Stackelberg theory* [7] which studies games of selfish player, but where a fraction of the entire population is orchestrated by a global leader. However, in contrast to our paper, the leader is not bound to offer any incentives (or credits) to follow his objectives. Moreover, in the recent research thread of *combinatorial agencies* [1], a setting is studied where a mechanism designer seeks to influence the outcome of a game by contracting the players individually; however, he is not able to observe the player's individual actions but the contracts can only depend on the overall outcome.

III. MODEL

This section first reviews some basic definitions and formalisms from game theory which are needed for our analysis. Subsequently, the concept of mechanism design with creditability is introduced.

A. Game Theory

A *strategic game* can be described by a tuple $G = (N, X, U)$. $N = \{1, 2, \ldots, n\}$ is the set of *players* and each player $i \in N$ can choose a *strategy* (action) from the set X_i. The product of all the individual players' strategies is denoted by $X := X_1 \times X_2 \times \ldots \times X_n$. In the following, a particular outcome $x \in X$ is called *strategy profile* and we refer to the set of all other players' strategies of a given player i by $X_{-i} = X_1 \times \ldots \times X_{i-1} \times X_{i+1} \times \ldots \times X_n$. An element of X_i is denoted by x_i, and similarly, $x_{-i} \in X_{-i}$; hence x_{-i} is a vector consisting of the strategy profiles of x_i. Finally, $U = (U_1, U_2, \ldots, U_n)$ is an n-tuple of *payoff functions*, where $U_i : X \mapsto \mathbb{R}$ determines player i's payoff arising from the game's outcome.

The *social gain* of a game's outcome is given by the sum of the individual players' payoffs at the corresponding strategy profile x, i.e. $gain(x) := \sum_{i=1}^{n} U_i(x)$. Let $x_i, x'_i \in X_i$ be two strategies available to player i. We say that x_i *dominates* x'_i iff $U_i(x_i, x_{-i}) \geq U_i(x'_i, x_{-i})$ for every $x_{-i} \in X_{-i}$ and there exists at least one x_{-i} for which a strict inequality holds. x_i is the *dominant* strategy for player i if it dominates every other strategy $x'_i \in X_i \backslash \{x_i\}$. x_i is a *non-dominated* strategy if no other strategy dominates it. By $X^* = X_1^* \times X_2^* \times \ldots \times X_n^*$ we will denote the set of non-dominated strategy profiles, where X_i^* is the set of non-dominated strategies available to the individual player i.

The set of *best responses* $B_i(x_{-i})$ for player i given the other players' actions is defined as $B_i(x_{-i}) := \{\arg\max_{x_i \in X_i} U_i(x_i, x_{-i})\}$. A *Nash equilibrium* is a strategy profile $x \in X$ such that for all $i \in N$, $x_i \in B_i(x_{-i})$.

B. k-Implementation

We use the standard assumption that players are rational and always choose a non-dominated strategy. We examine the impact of payments to players offered by a *mechanism designer* (an interested third party) who seeks to influence the outcome of a game. These payments $V = (V_1, V_2, \ldots, V_n)$ are described by a tuple of non-negative payoff functions, where $V_i : X \mapsto \Re$, i.e. they depend on the strategies player i selects as well as on the choices of all other players. The original game $G = (N, X, U)$ is modified to $G_V := (N, X, [U + V])$ by these payments, where $[U + V]_i(x) = U_i(x) + V_i(x)$, hence each player i obtains the payoff of V_i in addition to the payoffs of U_i. The players' choice of strategies changes accordingly: Each player now selects a non-dominated strategy in G_V. Henceforth, the set of non-dominated strategy profiles of G_V is denoted by $X^*(V)$. A *strategy profile set* — also called *strategy profile region* — $O \subseteq X$ of G is a subset of all strategy profiles X, i.e., a region in the payoff matrix consisting of one or multiple strategy profiles. Similarly to X_i and X_{-i}, we define $O_i := \{x_i | \exists x_{-i} \in X_{-i} \text{ s.t. } (x_i, x_{-i}) \in O\}$ and $O_{-i} := \{x_{-i} | \exists x_i \in X_i \text{ s.t. } (x_i, x_{-i}) \in O\}$.

The mechanism designer's main objective is to force the players to choose a certain strategy profile or a set of strategy profiles, without having to spend too much. Concretely, for a desired strategy profile region O, we say that payments V *implement* O if $\emptyset \subset X^*(V) \subseteq O$. V is called *k-implementation* if, in addition $\sum_{i=1}^{n} V_i(x) \leq k \ \forall x \in X^*(V)$. That is, the players' non-dominated strategies are within the desired strategy profile, and the payments do not exceed k for any possible outcome. Moreover, V is an *exact k-implementation* of O if $X^*(V) = O$ and $\sum_{i=1}^{n} V_i(x) \leq k \ \forall x \in X^*(V)$. The *costs* $k(O)$ of implementing O is the lowest of all non-negative numbers q for which there exists a q-implementation. If an implementation meets this lower bound, it is optimal, i.e., V is an *optimal implementation* of O if V implements O and $\max_{x \in X^*(V)} \sum_{i=1}^{n} V_i(x) = k(O)$. The cost $k^*(O)$ of implementing O exactly is the smallest non-negative numbers q for which there exists an exact q-implementation of O. V is an *optimal exact implementation* of O if it implements O exactly and requires the costs $k^*(O)$. The set of all implementations of O will be denoted by $\mathcal{V}(O)$, and the set of all exact implementations of O by $\mathcal{V}^*(O)$. Finally, a strategy profile region $O = \{z\}$ of cardinality one — consisting of only one strategy profile — is called a *singleton*. For simplicity's sake we often write z instead of $\{z\}$ and $V(z)$ instead of $\sum_{i \in N} V_i(z)$. Clearly it holds for singletons that non-exact and exact k-implementations are equivalent. Moreover, observe, that only subsets of X which are in $2^{X_1} \times 2^{X_2} \times \ldots \times 2^{X_n} \subset 2^{X_1 \times X_2 \times \ldots \times X_n}$ can be implemented exactly. We call such a subset of X a *convex strategy profile region*.[2]

C. (Byzantine) Potential

k-implementation provides a tool for mechanism designers to implement desired outcomes in games. However, the question remains for which games it makes sense to take influence at all and if it does, which outcome is worth implementing. For that matter we look at two diametrically opposed kinds of interested parties. One being benevolent towards the participants of the game, the other being malicious. We presume that the former is interested in increasing a game's social gain whereas the latter wants the players' welfare to be as low as possible, preferably negative! For both, we define a measure aimed to indicate whether the mechanism of implementation enables them to modify a game in such a favorable way that their gain exeeds the manipulation's cost. We call this measure *potential* and *Byzantine potential* respectively. We define a single strategy profile's potential first.

Definition 3.1 ((Byzantine) Singleton Potential): A strategy profile $z \in X$ has a *singleton potential*

$$POT(z) := gain(z) - \max_{x^* \in X^*} gain(x^*) - k(z)$$

and a *Byzantine singleton potential*

$$BPOT(z) := \min_{x^* \in X^*} gain(x^*) - gain(z) - k(z) \ .$$

If a singleton z has a (Byzantine) potential of p, this means that by implementing z, a (Byzantine) interested party gains p more than she pays. If z's (Byzantine) potential is larger than 0, this indicates that z is worth implementing for the respective interested party.

Note that if the original game has several non-dominated outcomes ($|X^*| > 1$) we assume the worst case, namely that the players would choose, the non-dominated strategy profile with maximal gain for POT and the one with minimal gain for $BPOT$.

As it is not only possible to implement singletons but also sets of strategy profiles, we define a potential for sets as well.

Definition 3.2 ((Byzantine) Set Potential): A strategy profile region O has a *potential*

$$POT(O) := \max_{V \in \mathcal{V}(O)} \left[\min_{z \in X^*(V)} \left[gain(z) - V(z)\right]\right] - \max_{x^* \in X^*} gain(x^*)$$

and a *Byzantine potential*

$$BPOT(O) := \min_{x^* \in X^*} gain(x^*) - \min_{V \in \mathcal{V}(O)} \left[\max_{z \in X^*(V)} \left[gain(z) + V(z)\right]\right].$$

Note that we slightly misuse notation by reusing the $POT-$, $BPOT-$Function for singletons, strategy profile regions and entire games.

Definition 3.3 ((Byzantine) Game Potential): A game $G = (N, X, U)$ has a *potential* $POT(G) := \max_{O \subseteq X} POT(O)$ and a *Byzantine* potential $BPOT(G) := \max_{O \subseteq X} BPOT(O)$
We sometimes refer to a game G with $POT(G) \leq 0$ ($BPOT(G) \leq 0$) as a game which has no (Byzantine) potential.

Similar to the concept of exact implementation, we define an *exact potential* $POT^*(O)$ and an *exact Byzantine potential* $BPOT^*(O)$ by simply adapting $\mathcal{V}(O)$ to $\mathcal{V}^*(O)$ in the definition of $POT(O)$ and $BPOT(O)$. Thus, exact (Byzantine) potential measures a region's potential if the interested party may only use payments V which implement O exactly.

[2]because these regions define a convex area in the payoff matrix, provided that the strategies are depicted such that all o_i are next to each other.

IV. Standard Potential

This section studies potential and Byzantine potential in games. We start the analysis with singleton (Byzantine) potential and then proceed to the potential of entire sets. Finally, we state an exponential correct algorithm for finding a strategy profile region's (Byzantine) potential and present polynomial approximation algorithms.

A. Singletons

Consider an interested party who has to be sure about which specific strategy profile will rise from the game. More formally speaking, we presume her to be interested only in implementing singleton regions. Recall the formula for singleton potential, $POT(z) = gain(z) - \max_{x^* \in X^*} gain(x^*) - k(z)$. As Monderer and Tennenholtz found in [5], there is a polynomial formula for a singleton's cost, $k(z) = \sum_{i=1}^{n} \max_{x_i \in X_i} (U_i(x_i, z_{-i}) - U_i(z_i, z_{-i}))$. Thus the complexity of computing singleton potential (and Byzantine singleton potential) is polynomial. A logical task for the benevolent mechanism designer would be to find a game's best singleton, i.e. the strategy profile with the highest singleton potential. Whereas the dual task for a malicious designer would be to find the profile with the highest Byzantine potential. We propose an algorithm that computes two arrays, POT and $BPOT$, containing all (Byzantine) singleton potentials within a strategy profile region O. By setting $O = X$, the algorithm computes all singleton (Byzantine) potentials of an entire game.

Algorithm 1 initializes the POT-array with the negative value of the original game's maximal social gain in the non-dominated region and the $BPOT$-array with its minimal social gain. In the remainder, the algorithm then, for each player and strategy profile, adds up the player's contribution to the profiles' (Byzantine) potential. In any field z of the potential array POT, we add the amount that player i would contribute to the social gain if z was played and subtract the cost we had to pay him, namely $U_i(x_i, x_{-i}) - (m - U_i(x_i, x_{-i})) = 2U_i(x_i, x_{-i}) - m$. For any entry z in the Byzantine potential array $BPOT$, we subtract player i's contribution to the social gain and also the amount we would have to pay if z was played, $-U_i(x_i, x_{-i}) - (m - U_i(x_i, x_{-i})) = -m$. At the end of the algorithm, POT and $BPOT$ contain all singleton potentials, resp. singleton Byzantine potentials in O. The interested party may then find the best singleton by searching the maximal entry in the respective array which takes $O(|O|)$ time.

Algorithm 1 Compute Singleton (Byzantine) Potentials

```
Input: Game $G$, Region $O \subseteq X$
Output: Matrices $POT$ and $BPOT$
1: for all strategy profiles $x \in O$ do
2:     $POT[x] := -\max_{x^* \in X^*} gain(x^*)$;
3:     $BPOT[x] := \min_{x^* \in X^*} gain(x^*)$;
4: for all players $i \in N$ do
5:     for all $x_{-i} \in O_{-i}$ do
6:         $m := \max_{x_i \in X_i} U_i(x_i, x_{-i})$;
7:         for all strategies $z_i \in O_i$ do
8:             $POT[z_i, x_{-i}]$ += $2 * U_i(z_i, x_{-i}) - m$;
9:             $BPOT[z_i, x_{-i}]$ -= $m$;
```

Theorem 4.1: For a game where every player has at least two strategies, Algorithm 1 computes all singleton (Byzantine) potentials within a strategy profile region O in $O(n\,|X|)$ time.

PROOF. Algorithm 1 is correct because it applies the (Byzantine) singleton potential formula in order to compute POT and $BPOT$. Finding the non-dominated strategies in the original game is exactly the task the players do in order to determine their strategy. That is why we do not consider this task in the algorithm's runtime analysis. Finding the maximal, resp. minimal $gain$ amongst the possible outcomes X^* of the original game can be done in $O(n\,|X^*|)$. Computing m takes $O(|X_i|)$. Updating POT and $BPOT$ is done in constant time. Adding up over all foreach clauses, we get for this algorithm's runtime:$O\left(n\,|X^*| + \sum_{i \in N} \underbrace{[|O_{-i}| * (|X_i| + |O_i|)]}_{=|O_{-i}||X_i|+|O|} \right) = O(n\,|X^*| + n \max_{i \in N}(|O_{-i}|\,|X_i|) + n\,|O|) \in O(n\,|X|)$ □

When Algorithm 1 is used for computing all (Byzantine) singleton potentials throughout the game, i.e. $O = X$, the input consists of $|X|$ strategy profiles which each bear n independent payoffs (one for each player). Thus, the presented algorithm is linear with respect to its input.

Corollar 4.1: Algorithm 1 computes all (Byzantine) singleton potentials in linear time.

B. Sets

Implementing only singletons might often be of no disadvantage to the interested party, namely in games where the strategy profile region yielding the largest (Byzantine) potential is of cardinality 1. In general, however, in order to find a subregion of X yielding maximal (Byzantine) potential the designer must consider regions consisting of more than one strategy profile as well. We can construct games where the difference between the best (Byzantine) set potential and the best (Byzantine) singleton potential gets arbitrarily large. Figure 3 depicts such a game for the case of potential. A similar game

$G =$

20, 0	11, 9	10, 10	10, 10
11, 9	20, 0	10, 10	10, 10
19, 10	10, 19	9, 11	0, 20
10, 19	19, 10	0, 20	9, 11

$V =$

0, ∞	0, ∞	0, 0	0, 0
0, ∞	0, ∞	0, 0	0, 0
1, 1	1, 1	∞, 0	∞, 0
1, 1	1, 1	∞, 0	∞, 0

Fig. 3. 2-player Game where region O bears the largest potential. Implementation V yields $|X^*(V)| = O$. By offering payments V, a benevolent mechanism designer has cost 2, no matter which $o \in O$ will be played. However, she improves the social welfare by $(10 + 19) - (10 + 10) = 9$. Thus O has a potential of 7 whereas any singleton $o \in O$ has a potential of $29 - 20 - 11 = -2$. By reducing player 2's payoffs in the upper game half and player 1's payoffs in the right game half, O 's potential gets arbitrarily large.

can be used to show an arbitrarily large difference in Byzantine potentials: E.g. set the payoffs in the four upper right strategy profiles of the game G in Figure 3 to 100 instead of 10. V still implements O but switching to O now decreases the social gain tremendously.

Although many factors and parameters have an influence on a strategy profile region's (Byzantine) potential, there are a few observations one can make when looking at some examples. The first is the fact that if rational players already choose strategies such that the strategy profile with the highest social gain is non-dominated, a designer will not be able to ameliorate the game. Just as well, a malicious interested party will have nothing to corrupt if a game already yields the lowest social gain possible.

Fact 4.2:

(i) If a game G's social optimum $x_{opt} := \arg\max_{x \in X} gain(x)$ is in X^* then G has potential.

(ii) If a game G's social worst $x_{wrst} := \arg\min_{x \in X} gain(x)$ is in X^* then G has no Byzantine potential.

A class of games where both properties (i) and (ii) of Fact 4.2 always hold is the class of *equal sum games*. Equal sum games are games where every strategy profile yields the same gain, this is $gain(x) = c \ \forall x \in X, c :$ *constant*.

Fact 4.3 (Equal Sum Games): An equal sum game has neither potential nor Byzantine potential.

A well-known example of an equal sum game is *Matching Pennies*, depicted in Figure 4. This example game

$MP =$

1, -1	-1, 1
-1, 1	1, -1

Fig. 4. Matching Pennies. Both players toss a penny, if both show the same face, player 2 gives his penny to player 1, if the pennies do not match, player 2 gets the pennies. This game has neither potential nor Byzantine potential.

features another special property: There is no dominated strategy. Therefore an interested party could only implement strategy profile regions O which are subsets of X^*. This raises the question whether a region $O \subseteq X^*$ can ever have (Byzantine) potential. We find that the answer is no and moreover:

Theorem 4.4: A strategy profile region $O \subseteq X$ intersecting with the region of non-dominated strategy profiles X^* has neither potential nor Byzantine potential.

PROOF. Assume that $|O \cap X^*| > 0$ and let z be a strategy profile in the intersection of O and X^*. Let $x^*_{max} := \arg\max_{x^* \in X^*} gain(x^*)$ and $x^*_{min} := \arg\min_{x^* \in X^*} gain(x^*)$. We get for the potential $POT(O) = \min_{o \in O}\left[gain(o) - k^O(o)\right] - gain(x^*_{max}) \leq \left[gain(z) - k^O(z)\right] - gain(x^*_{max}) \leq gain(x^*_{max}) - k^O(z) - gain(x^*_{max}) = -k^O(z) \leq 0$, and for the Byzantine potential $BPOT(O) = gain(x^*_{min}) - \max_{o \in O}\left[gain(o) + k^O(o)\right] \leq gain(x^*_{min}) - \left[gain(z) + k^O(z)\right] \leq gain(z) - gain(z) - k^O(z) = -k^O(z) \leq 0$. $\square$

In general, though, the problem of computing a strategy profile region's (Byzantine) potential seems computationally hard. As it is very much related to the problem of computing a region's implementation cost, which is (unfortunately not proved, but) conjectured in [2] to be **NP**-hard, we also conjecture the problem of finding $POT(O)$ or $BPOT(O)$ to be **NP**-hard in general. We can in fact, show that computing Byzantine potential has at least the same complexity as computing a region's cost.

Theorem 4.5: The problem of computing a strategy profile region's Byzantine potential is at least as hard as the problem of computing the region's implementation cost.

PROOF. We show Theorem 4.5 by reducing the problem of computing $k(O)$ to the problem of computing

$BPOT(O)$. Theorem 4.4 allows us to assume O and X^* do not intersect since $|O \cap X^*|$ implies $BPOT(O) = 0$ anyway. A strategy profile region's cost are by definition $k(O) = \min_{V \in \mathcal{V}(O)} \left(\max_{z \in X^*(V)} V(z) \right)$ and from the Byzantine potential's definition (Definition 3.2), we have $\min_{V \in (V)} \left(\max_{z \in X^*(V)} [gain(z) + V(z)] \right) = \min_{x^* \in X^*} gain(x^*) - BPOT(O)$. We see that the latter equation's left hand side almost matches the formula for $k(O)$ if not for the term $gain(z)$. If we can manage to modify the given game such that all strategy profiles inside $X^*(V) \subseteq O$ have a gain of 0 without modifying O's cost, we will be able to reduce O's cost to $k(O) = \min_{x^* \in X^*} gain(x^*) - BPOT(O)$. This is achieved by the following transformation of a cost problem instance (G, O) into a $BPOT$ problem instance (G', O): Add an additional player $n+1$ with one strategy a and a payoff function $U_{n+1}(x)$ equal to $-gain(x)$ if $x \in O$ and 0 otherwise. Thus, a strategy profile x in G' has social gain equal to 0 if it is in O and equal to $gain(x)$ in the original game if it is outside O. As player $n+1$ has only one strategy available, G' has the same number of strategy profiles as G and furthermore, there will be no payments V_{n+1} needed in order to implement O. Player $(n+1)$'s payoffs impact only on the profiles' gain. They have no effect on how the other players decide their tactics. Thus, the non-dominated region in G' is the same as in G and it does not intersect with O. Since the transformation does not affect the term $\min_{x^* \in X^*} gain(x^*)$, the region's cost in G are equal to $\min_{x^* \in X^*} gain(x^*) - BPOT(O)$ in G'. □

C. Algorithms

The task of finding a strategy profile region's potential is computationally hard. Recall that we have to find an implementation V of O which maximizes the term $\min_{z \in X^*(V)} [gain(z) - V(z)]$. Thus, there is at least one implementation $V \in \mathcal{V}(O)$ bearing O's potential. Since this V implements a subregion of O exactly, it is also valid to compute O's potential by searching among all subregions O' of O the one with the largest exact potential $POT^*(O')$. [3]

In the following we provide an algorithm which computes a convex strategy profile region's exact potential. It makes use of the fact that if $X^*(V)$ is supposed to be a subset of O, each strategy $\bar{o}_i \notin O_i$ must be dominated by at least one strategy o_i in the resulting game $G(V)$. Algorithm 1 in [2] which computes a region's exact cost is based on the same property. We will not give an explicit algorithm for computing $POT(O)$ because we cannot provide a more efficient way to solve this problem if not by computing all convex subregions' exact potential with Algorithm 2 and returning the largest potential found.

[3]Note that we do not provide algorithms for computing Byzantine potential but rather potential only. This is not because we decide to support only benevolent designers but because we consider a villain – or a reader interested in how a villain might corrupt a game – capable of transforming the potential algorithms into their Byzantine versions.

Algorithm 2 Exact Potential

```
Input: Game $G$, convex region $O$ with $O_{-i} \subset X_{-i} \forall i$
Output: $POT^*(O)$
1: $V_i(x) := 0$, $W_i(x) := 0$ $\forall x \in X$ , $i \in N$;
2: $V_i(o_i, \bar{o}_{-i}) := \infty$ $\forall i \in N, o_i \in O_i, \bar{o}_{-i} \in X_{-i} \backslash O_{-i}$;
3: return ExactPOT($V$, 1)$- \max_{x^* \in X^*} gain(x^*)$;
ExactPOT(V, i):
Input: payments $V$, current player $i$
Output: $\max_{W \in \{W | W(x) \geq V(x)\ \forall x \in X\}} \min_{o \in O} (gain(o) - W(o))$
1: if $|X_i^*(V) \backslash O_i| > 0$ then
2:    $s :=$ any strategy in $X_i^*(V) \backslash O_i$; $pot_{best} := 0$;
3:    for all $o_i \in O_i$ do
4:       for all $o_{-i} \in O_{-i}$ do
5:          $W(o_i, o_{-i})$:=$\max\{0, U_i(s, o_{-i}) -$
            $(U_i(o_i, o_{-i}) + V(o_i, o_{-i}))\}$;
6:       $pot :=$ ExactPOT($V + W$, $i$);
7:       if $pot > pot_{best}$ then
8:          $pot_{best} := pot$;
9:       for all $o_{-i} \in O_{-i}$ do
10:         $W(o_i, o_{-i}) := 0$;
11:   return $pot_{best}$;
12: else if $i < n$ then
13:   return $ExactPOT(V, i + 1)$;
14: else
15:   return $\min_{o \in O} (gain(o) - V(o))$;
```

Theorem 4.6: Algorithm 2 computes a strategy profile region's exact potential in time $O\left(n\,|X| + n \left(\max_{i \in N} |O_i|^{n \max_{i \in N} |X_i^*|}\right)\right)$.

PROOF. The algorithm is correct because it recursively searches all possibilities of a strategy in $X_i \backslash O_i$ to be dominated by a strategy in O_i. For further explanations on correctness and a proof of the algorithm's runtime see the proof of Algorithm 1 in [2]. □

Note that Algorithm 2 has a non-polynomial time complexity which we presume to be inavoidable for a correct algorithm.

However, there are polynomial algorithms which approximate $POT(O)$, $POT^*(O)$, $BPOT(O)$ or

$BPOT^*(O)$. We present a number of algorithms which all provide a lower bound of $POT(O)$. A rather simple lower bound is found when searching for the maximal singleton (Byzantine) potential inside the target region. Therefore, Algorithm 1 provides one possibility of approximating a region's (Byzantine) potential. One has it compute all singleton (Byzantine) potentials inside O and then searches the returned arrays for the largest entry. Unfortunately, the game depicted in Figure 3 describes a game configuration which allows the difference between the best (Byzantine) singleton potential and the best (Byzantine) set potential to be arbitrarily large. Thus, a best singleton approximation is arbitrarily bad in general.

We present a greedy algortihm (cf Algorithm 3) which associates each strategy $x_i^* \in X_i^* \backslash O_i$ yet to be dominated with strategy $o_i \in O_i$ maximizing the term $\min_{o_{-i} \in O_{-i}}(gain(o_i, o_{-i}) - V(o_i, o_{-i}) - \max\{0, U_i(s, o_{-i}) - U_i(o_i, o_{-i}) - V(o_i, o_{-i})\})$. This is, in each step we choose to dominate a non-dominated strategy with a strategy such that in the worst case,i.e. if $X^*(V)$ gets to be the entire region O, the potential is maximized. Similarly to the SETCOVER greedy algorithm [3] which chooses in each step the subset covering the most elements not covered already, Algorithm 3 selects a pair of x_i^*,o_i such that by dominating x_i^* with o_i, the number of strategies in $X_i^* \backslash O_i$ that will be dominated therewith is maximal. Thus, in each step there will be an o_i assigned to dominate x_i^* which is locally worst case optimal. Additionally, if there is the chance of dominating multiple strategies, the algorithm takes it.

Theorem 4.7: Algorithm 3 returns a lower bound of $POT^*(O)$ in $O\left(n\,|X| + n\,|O| \max_{i \in N}\left[|X_i^* \backslash O_i|^3\,|O_{-i}|^2\right]\right)$.

PROOF. The algorithm is correct, since it finds a correct implementation V of O and computes the improvement V brings. Any improvement induced by a correct implementation is at most as large as O's potential. For a proof of the algorithm's runtime we refer to the proof of Algorithm 3 in [2]. □

Algorithm 3 Greedy Computation of Potential

```
Input: Game $G$, convex target region $O$
Output: Approximation(Lower Bound) of $POT(O)$
 1: $V_i(x) := 0; W_i(x) := 0 \ \forall x \in X \ , \ i \in N$;
 2: for all $i \in N$ do
 3:   $V_i(o_i, \bar{o}_{-i}) := \infty \ \forall o_i \in O_i \ , \ \bar{o}_{-i} \in X_{-i} \backslash O_{-i}$;
 4:   while $X_i^*(V) \not\subseteq O_i$ do
 5:     $c_{best} := 0; m_{best}$ :=null; $s_{best}$ :=null;
 6:     for all $s \in X_i^*(V) \backslash O_i$ do
 7:       $m := \arg\min_{o_i \in O_i} MinPot(s, o_i)$;
 8:       for all $o_{-i} \in O_{-i}$ do
 9:         $W(m, o_{-i})$:=$\max\{0, U_i(s, o_{-i}) - (U_i(m, o_{-i}) + V(m, o_{-i}))\}$;
10:       $c := 0$ ;
11:       for all $x \in X_i^*$ do
12:         if $m$ dominates $x$ in $G(V + W)$ then
13:           $c++$;
14:       if $c > c_{best}$ then
15:         $c_{best} := c$ ; $m_{best} := m$ ; $s_{best} := s$ ;
16:     for all $o_{-i} \in O_{-i}$ do
17:       $V(m_{best}, o_{-i})$+=$\max\{0, U_i(s_{best}, o_{-i}) - (U_i(m_{best}, o_{-i}) + V(m_{best}, o_{-i}))\}$;
18: return $\min_{x \in X^*(V)}[gain(x) - V(x)] - \max_{x^* \in X^*} gain(x^*)$;
MinPot(s, $o_i$):
 1: return $\min_{o_{-i} \in O_{-i}}(gain(o) - V(o) - \max\{0, U_i(s, o_{-i}) - U_i(o) - V(o)\})$;
```

V. Expected Implementation Cost

In the previous section, we always assumed the worst case to happen for the interested party. In particular, we defined the costs of an implementation V to be equal to the costs of the strategy profile in $X^*(V)$ with the highest payments. Now, this seems to be a point of view of a risk averse interested party. One might also think of an interested party which takes some risk and would like to influence the game even if in the worst case the costs are much higher than in the average case. In order to come up to a less risk averse mechanism designer's needs, we define the cost of an implementation V as the *average* of all strategy profiles' possible cost in $X^*(V)$. By doing so, we assume all non-dominated strategy profiles $x \in X^*(V)$ to have the same probability.

Definition 5.1 (Expected Cost): A strategy profile region O has *expected implementation cost* $k_{EXP}(O) := \min_{V \in \mathcal{V}(O)}\left[\varnothing_{z \in X^*(V)} V(z)\right]$ where $\varnothing$ is the average operator, meaning that $\varnothing_{x \in X} f(x) := \frac{1}{|X|} * \sum_{x \in X} f(x)$ and $V(z) := \sum_{i=1}^n V_i(z)$.

Similar to the worst case cost, we can define the expected cost for exactly implementing a strategy profile region as follows:

Definition 5.2 (Exact Expected Cost): A strategy profile region O has *exact expected implementation cost* $k^*_{EXP}(O) := \min_{V \in \mathcal{V}^*(O)}\left[\varnothing_{z \in X^*(V)} V(z)\right]$

A. Exact

Since the set of all exact implementations for a region O is a subset of all implementations for O, i.e.

$\mathcal{V}^*(O) \subseteq \mathcal{V}(O)$, it seems easier to have a look at exact implementations first.

Note that for strategy profile regions O with $k^*_{EXP}(O) = 0$ any exact implementation V must have zero payments for any profile inside O, i.e. $V_i(o) = 0 \forall\ i \in N, o \in O$, because otherwise, the interested party would risk paying something. Thus, for 0-implementable strategy profiles regions, the concepts of worst case exact cost and expected exact cost coincide, i.e., $k^*_{EXP}(O) = 0$ iff $k^*(O) = 0$. Therefore, Algorithm 4 decides not only if O has worst case exact cost of 0, but if it has expected exact cost of 0 as well. Corresponding to Theorem 4.2 in [2], the following holds:

Theorem 5.1: Algorithm 4 decides for a convex strategy profile region whether its expected implementation cost are equal to 0 or larger in $O(|X^*|\,|O|)$ time.

Algorithm 4 Expected Exact Zero Cost

```
Input: Game $G$, convex region $O$ with $O_{-i} \subset X_{-i} \forall i$
Output: true if $k^*_{EXP}(O) = 0$, false otherwise
 1: for all $i \in N$ do
 2:   for all $s \in X_i^* \backslash O_i$ do
 3:     dZero := false;
 4:     for all $o_i \in O_i$ do
 5:       b := true;
 6:       for all $o_{-i} \in O_{-i}$ do
 7:         b := b $\wedge$ ($U_i(s, o_{-i}) \leq U_i(o_i, o_{-i})$);
 8:       dZero := dZero $\vee$ b;
 9:     if !dZero then
10:       return false;
11: return true;
```

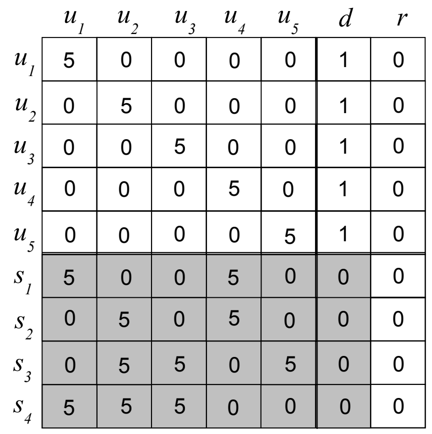
For cases where $k^*_{EXP}(O) > 0$, we can show that the problem of computing $k^*_{EXP}(O)$ is **NP**-hard.

Theorem 5.2: The problem of finding a strategy profile region's expected exact implementation cost is **NP**-hard unless $k^*_{EXP}(O) = 0$.

PROOF. In order to prove Theorem 5.2 we reduce the optimization version of the SETCOVER problem shown to be **NP**-hard by Karp in [3], to the problem of computing $k^*_{EXP}(O)$. The SETCOVER problem is the problem of finding, for a given universe $\mathcal{U}$ of n elements $\{e_1, e_2, \ldots, e_n\}$ and m subsets $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}, S_i \subset \mathcal{U}$ the minimal collection of S_i's which contains each element $e_i \in \mathcal{U}$. Karp proved SETCOVER to be **NP**-complete in his seminal work [3]. Given a SETCOVER problem instance $SC = (\mathcal{U}, \mathcal{S})$, we can construct a game $G = (N, X, U)$ where $N = \{1, 2\}$, $X_1 = \{u_1, u_2, \ldots, u_n, s_1, s_2, \ldots, s_m\}$, $X_2 = \{u_1, u_2, \ldots, u_n, d, r\}$. A strategy u_j correspond to element $e_j \in \mathcal{U}$, strategy s_j corresponds to a set S_j. Player 1's payoff function U_1 is defined as follows:

$$
\begin{aligned}
U_1(u_i, u_j) &:= \begin{cases} m+1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \\
U_1(s_i, u_j) &:= \begin{cases} m+1 & \text{if } e_j \in S_i \\ 0 & \text{otherwise} \end{cases} \\
U_1(u_i, d) &:= 1 \\
U_1(s_i, d) &:= 0 \\
U_1(x_1, r) &:= 0 \quad \forall x_1 \in X_1
\end{aligned}
$$

Player 2 has a payoff of 0 when she plays r and 1 otherwise. In this game, strategies u_j are not dominated for player 1 because in column d, $U_1(u_j, d) > U_1(s_i, d) \forall i \in \{1, \ldots m\}$. The region O we would like to implement is $\{(x_1, x_2) | x_1 = s_i \wedge (x_2 = u_i \vee x_2 = d)\}$. See Figure 5 for an example.



	u_1	u_2	u_3	u_4	u_5	d	r
u_1	5	0	0	0	0	1	0
u_2	0	5	0	0	0	1	0
u_3	0	0	5	0	0	1	0
u_4	0	0	0	5	0	1	0
u_5	0	0	0	0	5	1	0
s_1	5	0	0	5	0	0	0
s_2	0	5	0	5	0	0	0
s_3	0	5	5	0	5	0	0
s_4	5	5	5	0	0	0	0

Fig. 5. Payoff matrix for player 1 in game which reduces the SETCOVER problem instance $SC = (\mathcal{U}, \mathcal{S})$ where $\mathcal{U} = \{e_1, e_2, e_3, e_4, e_5\}$, $\mathcal{S} = \{S_1, S_2, S_3, S_4\}$, $S_1 = \{e_1, e_4\}, S_2 = \{e_2, e_4\}, S_3 = \{e_2, e_3, e_5\}, S_4 = \{e_1, e_2, e_3\}$ to the problem of computing $k^*_{EXP}(O)$. The optimal exact implementation V of O in this example game adds a payment V_1 of 1 to the strategy profile (s_1, d) and (s_3, d). This indicates that the two sets S_1 and S_3 cover $\mathcal{U}$ optimally.

We claim that if $Q = \{Q_1, Q_2, \ldots, Q_k\}$, where each Q_j corresponds to an S_i, is an optimal covering set for a SETCOVER problem, an optimal exact implementation V of O in the corresponsing game has payments

$$
\begin{aligned}
V_1(s_i, d) &:= \begin{cases} 1 & \text{if } Q_i \in Q \\ 0 & \text{otherwise} \end{cases} \\
V_1(s_i, u_j) &:= 0
\end{aligned}
$$

Such a payment matrix makes all strategies u_i of player 1 dominated since any strategy u_i representing element e_i is dominated by the strategies s_j corresponding to S_j which cover e_i in the minimal covering set.[4] If there are any strategies s_i dominated by other strategies s_j, we can make them non-dominated by setting the payments $V_1(s_i, r)$ in the extra column r accordingly.[5]

It remains to show that such an implementation is optimal indeed and there are no other optimal implementations not corresponding to a minimal covering set.

Note that by setting $V_1(s_i, d) = 1$ and $V_1(s_i, r) > 0$ for all s_i, all strategies u_j are guaranteed to be dominated and V implements O exactly with expected cost $\varnothing_{o\in O} V(o) = \frac{m}{|O|}$. If an implementation had a positive payment for any strategy profile of the form (s_i, u_j) it would have to be of an amount of at least $m+1$ to have an effect. A positive payment $\leq m+1$ would already yield larger cost, namely $\frac{m+1}{|O|}$, than the above mentioned implementation which sets all $V_1(s_i, d)$ to 1. Thus, an optimal V has positive payments inside region O only in column d. Due to our construction, by setting $V_1(s_i, d)$ to 1, s_i dominates the strategies u_j which correspond to the elements in S_i. An optimal implementation has a minimal number of 1's in column d. This can be achieved by selecting those strategies s_i, i.e. by setting $V_1(s_i, d) := 1$, which form a minimal covering set. □

B. General

Having proved that computing a strategy profile region O's expected exact implementation cost is $\mathbf{NP}$-hard, one expects the general problem of computing expected implementation cost to be $\mathbf{NP}$-hard as well. However, we need to come up with a rather sophisticated game configuration which reduces SETCOVER to the problem of finding an implementation of a strategy profile region with optimal expected cost. With this reduction, we can prove the following:

Theorem 5.3: The problem of finding a strategy profile region's expected implementation cost is $\mathbf{NP}$-hard.

PROOF. We give a similar reduction of the SETCOVER problem to the problem of computing $k_{EXP}(O)$ by extending the setup we used for proving Theorem 5.2. Given a SETCOVER problem instance $SC = (\mathcal{U}, \mathcal{S})$, we can construct a game $G = (N, X, U)$ where $N = \{1, 2, 3\}$, $X_1 = \{u_1, u_2, \ldots, u_n, s_1, s_2, \ldots, s_m\}$, $X_2 = \{u_1, u_2, \ldots, u_n, s_1, s_2, \ldots, s_m, d, r\}$, $X_3 = \{a, b\}$. Again, a strategy u_j corresponds to element $e_j \in \mathcal{U}$, strategy s_j corresponds to a set S_j. In the following, when we use "_" in profile vectors we thereby mean profiles where _ can be any of the current player's possible strategies. Player 1's payoff function U_1 is defined as follows:

$$\begin{aligned}
U_1(u_i, u_j, \_) &:= \begin{cases} (m+n)^2 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \\
U_1(u_i, s_j, \_) &:= 0 \\
U_1(s_i, u_j, \_) &:= \begin{cases} (m+n)^2 & \text{if } e_j \in S_i \\ 0 & \text{otherwise} \end{cases} \\
U_1(s_i, s_j, \_) &:= \begin{cases} 0 & \text{if } i = j \\ (m+n)^2 & \text{otherwise} \end{cases} \\
U_1(u_i, d, \_) &:= 1 \\
U_1(s_i, d, \_) &:= 0 \\
U_1(\_, r, \_) &:= 0
\end{aligned}$$

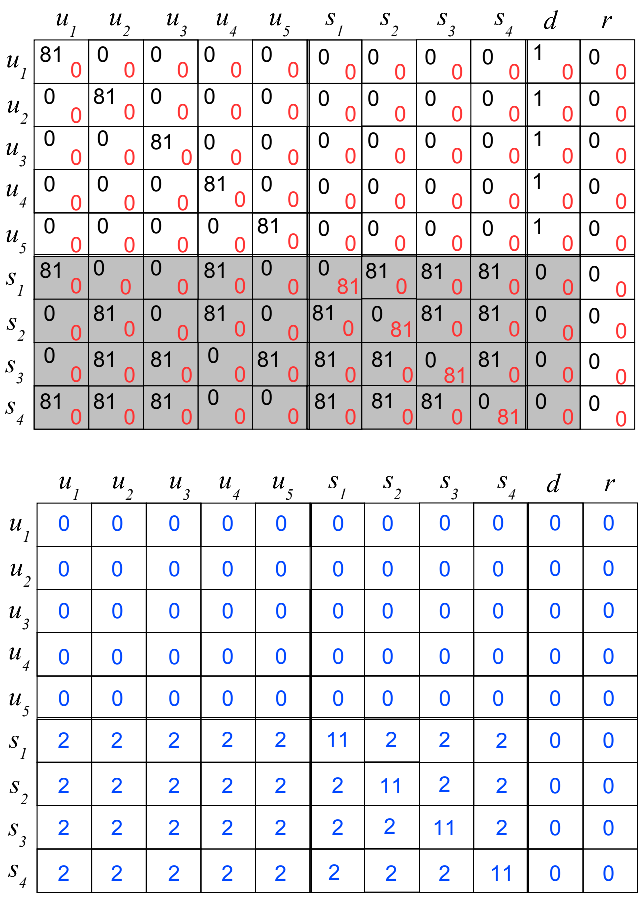
Player 2 has a payoff of $(m+n)^2$ for any strategy profile of the form $(s_i, s_i, \_)$, a payoff of 0 for when she plays r and 1 for any other strategy profile. Player 3 has a payoff of $(m+n)^2$ for strategy profiles of the form (s_i, s_i, b), a payoff of 2 for profiles (s_i, u_i, b) and profiles $(s_i, s_j, b), i \neq j$ and a payoff of 0 for any other profile. The region O we would like to implement is $\{(x_1, x_2, x_3) | x_1 = s_i$ and $(x_2 = u_i, x_2 = s_i$ or $x_2 = d)$ and $(x_3 = a)\}$. See Figure 6 for an example.

First, note the fact that any implementation of O will have $V_3(o_1, o_2, a) \geq U_3(o_1, o_2, b)$, in order to leave player 3 no advantage playing b instead of a. In fact, setting $V_3(o_1, o_2, a) = U_3(o_1, o_2, b)$ suffices.[6] Analogously to the proceeding in Theorem 5.2's proof, we claim that if $Q = \{Q_1, Q_2, \ldots, Q_k\}$, where each Q_j corresponds to an S_i, is an optimal covering set for a SETCOVER problem, an optimal exact implementation V of O in the corresponsing game selects a row s_i, i.e. it has payments $V_1(s_i, d, a) = 1$, if $Q_i \in Q$ and it does not select it $(V_1(s_i, d, a) = 0)$ otherwise. All other payments V_1 inside O are 0. Player 2's payments $V_2(o)$ are 0 for all $o \in O$ and player 3's payoffs are set $V_3(o_1, o_2, a) = U_3(o_1, o_2, b)$. A selected row s_i contributes expected cost of $cost_{s_i} = \frac{1}{n+m+1}(n+m)\cdot 2+n+m+1$. A non-selected row s_j contributes $cost_{s_j} = \frac{1}{n+m+1}(n+m)\cdot 2+n+m < cost_{s_i}$. Thus it is worth including also non-selected rows in $X^*(V)$. When selecting all rows s_i the expected cost are $cost_{allrows} = \varnothing_{i=1}^{m} cost_{s_i} = \frac{1}{n+m+1}3(n+m)+1 < 3$ In fact, the constructed game's payoffs are chosen such that it is not worth implementing any region smaller than

[4]If $|S_j| = 1$ s_j gives only equal payoffs in $G(V)$ to those of u_i in the range of O_2. However, s_j can be made dominating u_i easily by increasing s_j's payoff $V_1(s_j, r)$ in the extra column r.

[5]In [2], the proof of Theorem 5.1 shows that one column is enough to attain a game where no strategy dominates any other.

[6]Make it worthier for player 3 playing a in a profile outside O and thus convince her to choose a



	u_1	u_2	u_3	u_4	u_5	s_1	s_2	s_3	s_4	d	r
u_1	81, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	1, 0	0, 0
u_2	0, 0	81, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	1, 0	0, 0
u_3	0, 0	0, 0	81, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	1, 0	0, 0
u_4	0, 0	0, 0	0, 0	81, 0	0, 0	0, 0	0, 0	0, 0	0, 0	1, 0	0, 0
u_5	0, 0	0, 0	0, 0	0, 0	81, 0	0, 0	0, 0	0, 0	0, 0	1, 0	0, 0
s_1	81, 0	0, 0	0, 0	81, 0	0, 0	0, 81	81, 0	81, 0	81, 0	0, 0	0, 0
s_2	0, 0	81, 0	0, 0	81, 0	0, 0	81, 0	0, 81	81, 0	81, 0	0, 0	0, 0
s_3	0, 0	81, 0	81, 0	0, 0	81, 0	81, 0	81, 0	0, 81	81, 0	0, 0	0, 0
s_4	81, 0	81, 0	81, 0	0, 0	0, 0	81, 0	81, 0	81, 0	0, 81	0, 0	0, 0

	u_1	u_2	u_3	u_4	u_5	s_1	s_2	s_3	s_4	d	r
u_1	0	0	0	0	0	0	0	0	0	0	0
u_2	0	0	0	0	0	0	0	0	0	0	0
u_3	0	0	0	0	0	0	0	0	0	0	0
u_4	0	0	0	0	0	0	0	0	0	0	0
u_5	0	0	0	0	0	0	0	0	0	0	0
s_1	2	2	2	2	2	11	2	2	2	0	0
s_2	2	2	2	2	2	2	11	2	2	0	0
s_3	2	2	2	2	2	2	2	11	2	0	0
s_4	2	2	2	2	2	2	2	2	11	0	0

Fig. 6. Payoff matrix for player 1 and player 2 given player 3 choses a and payoff matrix for player 3 when she plays strategy b in game which reduces the SETCOVER problem instance $SC = (\mathcal{U}, \mathcal{S})$ where $\mathcal{U} = \{e_1, e_2, e_3, e_4, e_5\}$, $\mathcal{S} = \{S_1, S_2, S_3, S_4\}$, $S_1 = \{e_1, e_4\}, S_2 = \{e_2, e_4\}, S_3 = \{e_2, e_3, e_5\}, S_4 = \{e_1, e_2, e_3\}$ to the problem of computing $k_{EXP}(O)$. Every implementation V of O in a game thus constructed needs to add any positive payment in the second matrix to V_3, i.e. $V_3(x_1, x_2, a) = U_3(x_1, x_2, b)$, in order to convince player 3 of playing strategy a. An optimal implementation adds a payment V_1 of 1 to the strategy profile (s_1, d, a) and (s_3, d, a), indicating that the two sets S_1 and S_3 cover $\mathcal{U}$ optimally in the corresponding SETCOVER problem.

O. If an implementation V yields a subset $X^*(O) \subset O$ with a profile in row s_i, $X^*(V)$ must contain player 2's best response for this row, namely s_i, because otherwise, we would have to offer a payment of $U_2(s_i, s_i, a) = (n+m)^2$ which is already more than $cost_{allrows}$ and thus too much for V to be optimal. Hence, $X^*(V)$ must contain (s_i, s_i, a). However, if (s_i, s_i, a) is in $X^*(V)$ then $V_3(s_i, s_i, a) \geq U_3(s_i, s_i, b) = n + m + 2$ is part of the cost as well. The only way for a row s_i not to contribute too much to the cost is by compensating the costly profile (s_i, s_i) with other, cheaper profiles [7]. As all other profiles of this row cost only $V_3(s_i, \_, b) = 2$ – except for (s_i, d, a) which costs 1 or 0 – it makes no sense *not* to implement the whole row. Therefore, an optimal implementation yields $X^*(V) = O$ with the inalienable payments to player 3 and a minimal number of 1-payments to player 1 for strategy profiles (s_i, d, a) such that every u_j is dominated by at least one s_i. The number of 1-payments is minimal if the selected rows correspond to a minimal covering set. $\square$

VI. Expected Potential

How does it affect the (Byzantine) potential if we use the same model of an interested party doing average-case computation as in the previous section? Again, we assume the mechanism designer to be willing to take some risk if, in expectation, she improves the game, or worsens the game in case she is malicious. A such designer will offer payments yielding a non-dominated region which maximizes the *average* difference between social gain and disbursed payments. We introduce the concept of *expected (Byzantine) Potential.*

Definition 6.1 (Expected (Byzantine) Potential): A strategy profile region O has *expected Potential* $POT_{EXP}(O) := \max_{V \in \mathcal{V}(O)} \left[\varnothing_{z \in X^*(V)} \left(gain(z) - V(z)\right)\right] - \varnothing_{x^* \in X^*(V)}\, gain(x^*)$ and *expected Byzantine potential* $BPOT_{EXP}(O) := \varnothing_{x^* \in X^*(V)} \left[gain(x^*)\right] - \min_{V \in \mathcal{V}(O)} \left[\varnothing_{z \in X^*(V)} \left(gain(z) + V(z)\right)\right]$ where $\varnothing_{x \in X}\, f(x) := \frac{1}{|X|} \sum_{x \in X} f(x)$.
Note that the proposed interested party does not expect the worst case to happen in the original game either, but she rather calculates with the average social gain of the non-dominated region X^*.

A. Expected Potential and Worst-Case Potential

The point of view of a mechanism designer calculating with the average case is more optimistic than the one of a risk averse designer who ever presumes the worst case to happen. Thus the observation stating that the expected (Byzantine) potential is always at least as large as the normal (Byzantine) potential does not surprise.

Theorem 6.1: A region's expected (Byzantine) potential is always larger or equal the region's (Byzantine) potential, i.e. $POT_{EXP}(O) \geq POT(O)$ and $BPOT_{EXP}(O) \geq BPOT(O)$.

[7]Compensating here means to choose payments V such that the resulting $X^*(V)$ is extended with strategy profiles causing less cost than (s_i, s_i) and therefore, the implementation's average cost is diminished

PROOF. $POT_{EXP}(O) := \max_{V \in \mathcal{V}(O)}[\varnothing_{z \in X^*(V)} (gain(z) - V(z))] - \varnothing_{x^* \in X^*(V)} gain(x^*) \geq \max_{V \in \mathcal{V}(O)}[\min_{z \in X^*(V)} (gain(z) - V(z))] - \max_{x^* \in X^*(V)} gain(x^*) = POT(O)$.
$BPOT_{EXP}(O) := \varnothing_{x^* \in X^*(V)} [gain(x^*)] - \min_{V \in \mathcal{V}(O)}[\varnothing_{z \in X^*(V)} (gain(z) + V(z))] \geq \min_{x^* \in X^*(V)} [gain(x^*)] - \min_{V \in \mathcal{V}(O)}[\max_{z \in X^*(V)} (gain(z) + V(z))] = BPOT(O)$ □
Note that in the special case of expected singleton (Byzantine) potential, it holds that $POT_{EXP}(x) \leq POT(x)$ and $BPOT_{EXP}(x) \leq BPOT(x)$ where $x \in X$.

B. Exact

Analogous to the cost definition in the last section, we can define an *exact expected potential* POT^*_{EXP} and an *exact expected Byzantine potential* $BPOT^*_{EXP}$ by simply adapting $\mathcal{V}(O)$ to $\mathcal{V}^*(O)$ in the definition of POT_{EXP} and $BPOT_{EXP}$.

Theorem 6.2:
(i) Computing a strategy profile's exact expected potential is NP-hard.
(ii) Computing a strategy profile's exact expected Byzantine potential is NP-hard.

PROOF. In order to proof Theorem 6.2 we will show that when $POT^*_{EXP}(O)$ is found, we immediately find $k^*_{EXP}(O)$ which we already proved to be NP-hard in Theorem 5.2. This reduction is possible for the exact case only because any $z \in O$ is also in $X^*(V)$ for any $V \in \mathcal{V}^*(O)$. $POT_{EXP}(O) = \max_{V \in \mathcal{V}^*(O)} \left(\varnothing_{z \in X^*(V)} [gain(z) - V(z)]\right) - \varnothing_{z \in X^*(V)} [gain(x^*)] = \max_{V \in \mathcal{V}^*(O)} \left(\varnothing_{z \in X^*(V)} [gain(z)] - \varnothing_{z \in X^*(V)} [V(z)]\right) - \varnothing_{x^* \in X^*} [gain(x^*)] = \varnothing_{z \in X^*(V)} [gain(z)] - \min_{V \in \mathcal{V}^*(O)} \left(\varnothing_{z \in X^*(V)} [V(z)]\right) - \varnothing_{x^* \in X^*} [gain(x^*)] = \varnothing_{z \in X^*(V)} [gain(z)] - \mathbf{k^*_{EXP}(O)} - \varnothing_{x^* \in X^*} [gain(x^*)]$
$BPOT_{EXP}(O) := \varnothing_{x^* \in X^*} [gain(x^*)] - \min_{V \in \mathcal{V}^*(O)} \left(\varnothing_{z \in X^*(V)} [gain(z) + V(z)]\right) = \varnothing_{x^* \in X^*} [gain(x^*)] - \varnothing_{z \in X^*(V)} [gain(z)] - \min_{V \in \mathcal{V}^*(O)} \left(\varnothing_{z \in X^*(V)} [V(z)]\right) = \varnothing_{x^* \in X^*} [gain(x^*)] - \varnothing_{z \in X^*(V)} [gain(z)] - \mathbf{k^*_{EXP}(O)}$ □

C. General

The general problem of finding a strategy profile region's expected (Byzantine) potential seems harder or at least as hard as computing the exact (Byzantine) potential since $(B)POT^*_{EXP}(O)$ maximizes (minimizes) over all $V \in \mathcal{V}^*(0)$ and $(B)POT_{EXP}(O)$ over all $V \in \mathcal{V}(O)$ where $\mathcal{V}^*(0)$ is contained in $\mathcal{V}(0)$. Indeed we find that the following holds:

Theorem 6.3: The problem of computing a strategy profile region's expected Byzantine potential is **NP**-hard.

PROOF. Theorem 6.3 can be proved by reducing the **NP**-hard problem of computing $k_{EXP}(O)$ to the problem of computing $BPOT_{EXP}(O)$. This reduction uses the same polynomial game transformation as the proof of Theorem 4.5. By ensuring $gain(z) = 0 \ \forall z \in O$ in the transformed game we obtain a formula for $k_{EXP}(O)$ depending only on O's Byzantine potential and the average social gain, namely $k_{EXP}(O) = \varnothing_{x^* \in X^*} gain(x^*) - BPOT_{EXP}(O)$. Thus, computing $BPOT_{EXP}(O)$ computes $k_{EXP}(O)$ and therefore finding the expected Byzantine is **NP**-hard as well. □

D. Algorithms

To round off our analysis of expected (Byzantine) potential, we would like to provide the less risk averse mechanism designer with some useful algorithms. Recall the interested party in Section IV which only cares for implementing singletons. For her it does not matter whether she is pessimistic or optimistic in her assumptions about how the game turns out since in the case of implementing singletons the concepts of (Byzantine) potential and expected (Byzantine) potential coincide. She can therefore use Algorithm 1 without adapting anything to compute which singleton is an optimal means for her end.

A less risk averse benevolent mechanism designer with immense computational power might want to adapt Algorithm 2 which correctly computes $POT^*(O)$ in order to compute $POT^*_{EXP}(O)$. For that matter she only has to change '$\max$' in Line 3 to '$\varnothing$' and '$\min$' in Line 15 to '$\varnothing$'. She can then use this algorithm for computing $POT_{EXP}(O)$ if she calls it for any $O' \subseteq O$ and thus finds the subregion of O maximizing $POT^*_{EXP}(O')$.

Since the mentioned algorithm has exponential time complexity, we should also provide polynomial approximation algorithms. Once more, the simplest method to find a lower bound for $POT_{EXP}(O)$ is to search the singleton in O with the largest expected potential. Unfortunately, there are games (cf Figure 3) where this lower bound gets arbitrarily bad in the average analysis just as well as in the worst case analysis.

The greedy algorithm (Algorithm 3) can be adapted by changing '$\min$' in Line 18 and in the subroutine $MinPot(s, o_i)$ Line 1 and '$\max$' in Line 18 to '$\varnothing$'.

VII. Variations

As the mechanism of implementation offers many interesting extensions where the notion of potential and Byzantine potential is similarly applicable and useful, this section examines two additional models of rationality. If we assume that players do not just select *any* non-dominated strategy, but have other parameters influencing their decision process, our model has to be adjusted. Furthermore, in many (real world) games, players typically do not know which strategies the other players will choose. In this case, a player cannot do better than assume that the other players select a strategy at random. If a player wants to maximize her game under this assumption, she can take the *average payoff* of strategies into account. We study the consequences of this process of decision making. In a second part of this section, take a look at the dynamics of repeated games with an interested third party offering payments in each round.

A. Average Payoff

As a player may choose any non-dominated strategy, it is reasonable for a player i to compute the payoff which each of her strategy will bring *on average*. We assume that player i considers each combination of the other player's strategies $x_{-i} \in X_{-i}$ to be of the same likelihood. When determining her strategy, player i then computes what payoff each of her strategy will bring in expectation. Thus, each strategy x_i has an expected payoff of $p_i(x_i) := \frac{1}{|X_{-i}|}\sum_{x_{-i}\in X_{-i}} U_i(x_i, x_{-i})$ for player i. She will then choose the strategy $s \in X_i$ with the largest $p_i(s)$, i.e. $s = \arg\max_{s\in X_i} p_i(s)$. If multiple strategies have the same expected payoff, player i mixes these strategies and plays each uniformly at random. [8] For average payoff strategy games, we say that x_i *dominates* y_i iff $p_i(x_i) > p_i(y_i)$. Note that with this modified meaning of domination, the region of non-dominated strategies, X^*, differs just as well.

Theorem 6.1 in [2] has some convenient consequences on the concept of (Byzantine) potential in average payoff strategy games. Applying Definition 3.1 to average payoff strategy games, we get for the *average payoff singleton potential*

$$\begin{aligned} POT_{\varnothing}(z) &= gain(z) - \max_{x^*\in X^*} gain(x^*) - \underbrace{k(z)}_{=0} \\ &= gain(z) - \max_{x^*\in X^*} gain(x^*) \end{aligned}$$

[8]Note that this is *not* the same model of mixed strategy games used by many other works on game theory.

and for the *average payoff Byzantine singleton potential*

$$\begin{aligned} BPOT_{\varnothing}(z) &= \min_{x^*\in X^*} gain(x^*) - gain(z) - \underbrace{k(z)}_{=0} \\ &= \min_{x^*\in X^*} gain(x^*) - gain(z) \end{aligned}$$

Applying Definition 3.2, we get for the *average payoff set potential*

$$\begin{aligned} POT_{\varnothing}(O) &= \max_{V\in\mathcal{V}(O)} \left[\min_{z\in X^*(V)} [gain(z) - V(z)]\right] - \\ &\quad \max_{x^*\in X^*} gain(x^*) \\ &= \max_{V\in\mathcal{V}^M(O)} gain(x^*(V)) - \max_{x^*\in X^*} gain(x^*) \\ &= \max_{o\in O} gain(o) - \max_{x^*\in X^*} gain(x^*) \end{aligned}$$

and for the *average payoff Byzantine singleton potential*

$$\begin{aligned} BPOT_{\varnothing}(O) &= \min_{x^*\in X^*} gain(x^*) - \\ &\quad \min_{V\in\mathcal{V}(O)} \left[\max_{z\in X^*(V)} [gain(z) + V(z)]\right] \\ &= \min_{x^*\in X^*} gain(x^*) - \\ &\quad \min_{V\in\mathcal{V}^M(O)} gain(x^*(V)) \\ &= \min_{x^*\in X^*} gain(x^*) - \min_{o\in O} gain(o)\ , \end{aligned}$$

where $\mathcal{V}^M(O)$ is the set of all implementations V of O with $|X^*(V)| = 1$ and where $x^*(V)$ is such an implementation's unique non-dominated strategy profile.

To see that these simplifications are valid, one must realize that when computing the (Byzantine) potential of a strategy profile region, for any $V \in \mathcal{V}(O)$, there is only one strategy profile which actually determines V's (Byzantine) potential. Let V_{opt} be an implementation of O which bears O's optimal (Byzantine) potential. Let o_{picked} be the strategy profile in O for which $gain(o_{picked}) \mp V_{opt}(o_{picked})$ is minimal (maximal). Since in average payoff strategy games, every singleton is 0-implementable, by simply implementing the singleton o_{picked}, O's (Byzantine) potential is attained. Thus, there is an implementation $V \in \mathcal{V}^M(O)$ which reaches O's (Byzantine) potential and $\{o_{picked}\} = X^*(V)$, $V(o_{picked}) = 0$.

Fact 7.1:

(i) If a game's social maximum is outside X^* then its average payoff game potential is > 0. Moreover, $POT(G) = gain(x_{socialOpt}) - \max_{x^*\in X^*} gain(x^*)$, where $x_{socialOpt} = \arg\max_{x\in X} gain(x)$

(ii) If a game's social worst is outside X^* then its average payoff Byzantine game potential is > 0. Moreover, $BPOT(G) = \min_{x^* \in X^*} gain(x^*) - gain(x_{socialWorst})$, where $x_{socialWorst} = \arg\min_{x \in X} gain(x)$.

Fact 7.2: For a game $G = (N, X, U)$ and a strategy profile region $O \subseteq X$, it holds that

(i) $POT_{\varnothing}(O) \geq POT(O)$, $POT_{\varnothing}(G) \geq POT(G)$

(ii) $BPOT_{\varnothing}(O) \geq BPOT(O)$, $BPOT_{\varnothing}(G) \geq BPOT(G)$.

B. Round-based Mechanisms

We extend our analysis to dynamic, round-based games, where in each turn, one player can change his strategy. The interested party may offer payments to the current player if she chooses a certain strategy and thus influence her decision. Among many imaginable models we present a dynamic game which is comparable to the real world example of stock exchange, where the players hold and trade stocks, ever trying to increase their portfolio's valuation. In this model, the current strategy profile represents the current market situation – indicated by the actual stock quotation – in which the participants find themselves. We then think of a mechanism designer as a broker who has access to insider information. She takes advantage of this knowledge by reassuring her stockholding customers that if in the next round, they want to stop engaging her as their broker, she will pay them a certain amount of money depending on what strategy the chose. If, however, they would like to continue their collaboration, the broker's offer expires and she reoffers new payments.

More formally, we presume to find the game in an initial starting configuration $c^{T=1} = (s^0, p^1)$ where $s^0 \in X$ is the initial state and $p^1 \in N$ the player who is to choose a strategy this turn. Note that in state s^t, each player i only sees the states she can reach by changing her strategy given the other players remain with their chosen strategies. This is, in round t, player i sees only strategy profiles in $X^t_{visible,i} = X_i \times \{s^t_{-i}\}$. In every round t, the mechanism designer offers player p^t insurances V^t for the strategy profiles she can see. We assume the smallest additional payment the mechanism designer can offer to be $\epsilon > 0$. Player p^t then adds up payments V^t to the game's static payoff matrix U and chooses a strategy with the largest resulting payoff which is actually a best response to the current situation $B_{p^t}(s^t_{-p^t})$ in $G(V^t)$. Next is player $p^{t+1} = p^t \pmod n + 1$ to choose a strategy. We assume the players to be quite loyal towards the interested party such that they only disengage from her if they realize they are stuck in a configuration, i.e. $c^{T=t+n} = c^{T=t}$, or stuck in a sequence of configurations, i.e. $c^{T=t+\alpha n} = c^{T=t}, \alpha \in \mathbb{N}$. If a player i finds herself in a situation in which she has already been before, she lets the interested party offer her insurance once more, decides for a best response strategy and immediately collects the payoff $U_i(s^{t+1})$ and the insurance $V^t_i(s^{t+1})$. Subsequently, all other players make a final decision and quit the game as well.

Thus, a mechanism designer can guide the players to a desired strategy profile or an entire strategy profile region without disbursing any payments. In order to keep them inside the desired region, however, she has to eventually disburse the offered payments. We call the sequence of payments $V^T = V^0 V^1 \ldots$ *dynamic implementation* of O if by sequentially offering these payoffs, all players encash their payoffs at a time when the game is in a state inside O. A dynamic implementation V^T yields a *social gain* $gain(V^T) := \sum_{i \in N} U_i(s^{t(i)})$ and *implementation cost* $cost(V^T) := \sum_{i \in N} V^{t(i)}(s^{t(i)+1})$, where $t(i)$ is the round in which player i encashes her payoff. A strategy profile region O has *dynamic cost* $k_{dyn}(O) := \min_{V^T \in \mathcal{V}_{dyn}(O)} cost(V^T)$ where $\mathcal{V}_{dyn}(O)$ denotes the set of all dynamic implementations of O. Further, a region O has *dynamic potential* $POT_{dyn}(O) := \max_{\{V^T \in \mathcal{V}_{dyn}(O)\}}[gain(V^T) - cost(V^T) - \max_{x^* \in X^*} gain(x^*)]$ and *Byzantine dynamic potential* $BPOT_{dyn}(O) := \max_{\{V^T \in \mathcal{V}_{dyn}(O)\}}[\min_{x^* \in X^*} gain(x^*) - gain(V^T) - cost(V^T)]$

Theorem 7.3: A single strategy profile $z \in X$ can be dynamically implemented from a random starting configuration in $3n$ rounds with cost equal to $k_{dyn}(z)$.

PROOF. We prove Theorem 7.3 by introducing a dynamic implementation which always implements z in $3n$ rounds at $k_{dyn}(z)$ cost. Figure 7 depicts such an implementation for a two-player game.

Let the initial configuration $c^{T=1} = (s^0, 1)$ and $z \in X$ the desired strategy profile. In a first phase, the interested party offers payments V^t such that the current state $s^t = (s^t_t, s^t_{-t})$ changes to (z_t, s^t_{-t}). This can be achieved by setting $V^t(z_t, s^t_{-t})$ large enough, e.g. $V^t(z_t, s^t_{-t}) = \infty$. Thus, in round $T = n$, the payments V^n guide player n to choosing strategy z_n and the game gets to state z. Phase two starts with the configuration $c^{T=n+1} = (z, 1)$. Now, the mechanism designer chooses payments such that the game remains in z, i.e. $V^t(z) = \max_{x_{p^t} \in X_{p^t}}(U_{p^t}(x_{p^t}, z_{-p^t}) - U_{p^t}(z)) + \epsilon$ where $\epsilon > 0$ and $V^t(x) = 0$ for $x \neq z$. Round $2n+1$ is the beginning of phase three, when player 1 realizes that the game is in the same configuration as in round $n+1$. The interested party offers the same payments as in round $n + 1$,

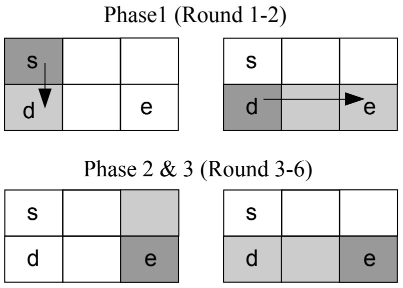


Fig. 7. Dynamic Implementation of strategy profile e starting from s. The regions visible to the current player are colored gray. The dynamic mechanism designer finds this two-player game in a starting configuration $c^{T=1} = (s, 1)$, i.e. the currently played strategy profile is s and player 1 is next to choose a strategy. The interested party now offers player 1 payments V^1 such that it is rational for her to change her strategy to e_1. Thus the game arrives at state $s^1 = d$. In the second round, it is player 2's turn. She is offered payments V^2 – which are forged such that e_2 is player 2's unique best response to d_1 in the game $G(V^2)$ – she decides to quit playing strategy s_2 and instead chooses e_2. The game arrives at profile e. Now the second phase begins. In round 3, the interested party convinces player 1 with payments V^3 to stay in e. She does the same for player 2 in round 4. In the third phase, starting with round 5, the game arrives at the same configuration as in round 3 and therefore player 1 chooses her strategy to be e_1 once more, has the interested party disburse the offered payments $V^5(e)$ and quits the game. Player 2 follows in round 6.

player 1 chooses to stay with z and encashes $U_1(z) + V^{2n+1}(z) = U_1(z) + \max_{x_1 \in X_1}(U_1(x_1, z_{-1}) - U_1(z)) + \epsilon$. In the next round, the interested party offers payments $V^{2n+2} = V^{n+2}$ and player 2 gets $U_2(z) + V^{2n+2}(z) = U_2(z) + \max_{x_2 \in X_2}(U_2(x_2, z_{-2}) - U_2(z)) + \epsilon$. The game ends when in round $3n$, player n is the last to cash in $U_n(z) + V^{3n}(z)$. Note that this scenario describes an optimal dynamic implementation of z if the start profile s^0 is $\neq z$. If the game starts at z, the mechanism designer can immediately begin phase two. It remains to show that this implementation's cost $cost(V^T)$ equals z's dynamic cost $k_{dyn}(z)$. Let us sum up all payments a mechanism designer disburses when implementing z as described. This is $\sum_{i=1}^{n} \max_{x_i \in X_i}(U_i(x_i, z_{-i}) - U_i(z)) + \epsilon$. No dynamic implementation of z can have smaller cost since in order to implement z, in round $t(i)$, the game configuration must be $c^{T=t(i)} = ((x_i, z_{-i}), i)$ and $V^{t(i)}$ must be such that strategy z_i is a unique best response to z_{-i} for player i. This can only be achieved by offering a payment $V^{t(i)}(z)$ of at least $\max_{x_i \in X_i}(U_i(x_i, z_{-i}) - U_i(z)) + \epsilon$ where ϵ can be 0 if z_i is the unique best response anyway. Therefore, the described dynamic implementation's cost are optimal. □

The dynamic implementation we just explained in Theorem 7.3's proof gives us a concrete formula for single strategy profiles's dynamic cost.

Theorem 7.4: A singleton's dynamic implementation cost are $k_{dyn}(x) = \sum_{i=1}^{n} \max_{x_i \in X_i}(U_i(x_i, z_{-i}) - U_i(z)) + \epsilon$

PROOF. Theorem 7.3's proof shows that each dynamic implementation $V^T \in \mathcal{V}_{dyn}(z)$ needs to make payments of at least $\sum_{i=1}^{n} \max_{x_i \in X_i}(U_i(x_i, z_{-i}) - U_i(z)) + \epsilon$. The given scenario also shows that this amount is sufficient for any $x \in X$ to be implemented dynamically. □

Theorem 7.4 points to an interesting observation one makes when comparing dynamic implementation mechanisms to static implementation. We find that the dynamic mechanism designer has to spend only ϵ more on implementing a singleton region than his static counterpart.

Corollar 7.1: A singleton's dynamic implementation cost $k_{dyn}(x)$ is equal to its static implementation cost $k(x)$ plus ϵ.

Therefore, a singleton has almost equal static and dynamic (Byzantine) potential.

Corollar 7.2:
(i) $POT_{dyn}(x) = POT(x) + \epsilon$
(ii) $BPOT_{dyn}(x) = BPOT(x) + \epsilon$

If a dynamic mechanism designer does not need all players to end up in the same strategy profile, but rather in a target region, she might profit from the fact that there is only one player choosing her strategy each round. The game might come by a strategy profile from where a certain player might see a very profitable state to her but she cannot get there because another player may choose first and thus makes this state invisible again. Note that for the cost of an implementation only the last n payments are taken into account. For a dynamic implementation of O, the last n states must be in O. Knowing the last n configurations in an optimal implementation is sufficient to derive the payments disbursed, since in order to have player i choose a strategy o_i given o_{-i} she must be offered a payment of $\max_{x_i \in X_i} U_i(x_i, o_{-i}) - U_i(o)$. Therefore the problem of finding an optimal dynamic implementation of O coincides with the problem of finding a sequence of n game configurations $c = (x, i)$ where $x \in O$. There are $n\,|O|\,(\max_{i \in N} |O_i|)^{n-1}$ possible sequences as there are $n\,|O|$ possible starting configurations and after that, each round, the mechanism designer has $|O_i|$ possibilities to guide the player to. We immediately see that if the number of players is bounded, one can compute a region's dynamic cost and its (Byzantine) dynamic potential in polynomial time.

VIII. Conclusion

Today's computer systems can no longer be assumed to consist of perfectly collaborating participants. Rather, different stakeholders may be involved who aim at maximizing their individual profits, and some players may even be malicious. This paper has introduced the study of the effect of a benevolent or malicious contractor whose goal is to change the game's outcome if the improvement (or worsening) in social welfare exceeds the cost. We have presented several insights for this problem and we have provided concrete algorithms for a worst-case model as well as for an average-case model. Additionally we have applied the concept of potential to games where players maximize their average payoff and to a dynamic game model. Our models still pose many interesting questions which have to be tackled in future research, including the quest for better approximation algorithms or for game classes which allow better potential approximation algorithms. Furthermore, we believe that the concept of potential and Byzantine potential shall be applied and analyzed in many other research areas dealing with agent based systems.

References

[1] M. Babaioff, M. Feldman, and N. Nisan. Combinatorial Agency. In *Proc. 7th ACM Conference on Electronic Commerce (EC)*, pages 18–28, 2006.

[2] R. Eidenbenz, Y. A. Oswald, S. Schmid, and R. Wattenhofer. Mechanism Design by Creditability. In *TIK Report 270, ETH Zurich*, 2007.

[3] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[4] Mas-Collel, Winston, and Green. *Microeconomic Thoery*. Oxford University Press, 1995.

[5] D. Monderer and M. Tennenholtz. k-Implementation. In *Proc. 4th ACM Conference on Electronic Commerce (EC)*, pages 19–28, 2003.

[6] Osborne and Rubinstein. *A Course in game Theory*. MIT Press, 1994.

[7] T. Roughgarden. Stackelberg Scheduling Strategies. In *Proc. ACM Symposium on Theory of Computing (STOC)*, pages 104–113, 2001.

[8] I. Segal. Contracting with Externalities. *The Quarterly Journal of Exonomics*, pages 337–388, 1999.