Ben-Gurion University of the Negev

Evolutionary Computation Project

Spring, 2015

A Genetic Algorithm for Efficient Equilibria in the “Best-Shot” Public Goods Game

Marina Simakov, 313113276

Kosta Slobodkin, 307179978

Zohar Komarovsky, 301812608

TABLE OF CONTENTS

1. Introduction

The “best-shot” public goods game is a network game, defined on a social network (Hirshleifer, 1983), (Jackson & Zenou, Games on networks, 2014). In network games the utilities of players depend both on their own actions and on the actions taken by their neighbors in the network. In the “best-shot” public goods game players share common goods and each player chooses whether to take some action or to avoid it. The action is associated with an investment in some local public good. The action might be in the form of performing some computational effort, where computation results can be easily shared locally. It can also be buying a book or some other product that is easily lent from one player to another. Each player wants to have at least one player in her neighborhood taking the action, including herself. However, there is a cost associated with taking the action, so if any of the player's neighbors takes the action then the player would prefer to avoid it.

Typically, there are many Pure-strategy Nash Equilibria (PNEs) in a “best-shot” game, each corresponds to a maximal independent set of vertices in the network (Bramoullé & Kranton, 2007). We refer to these outcomes as *stable*. Finding a maximal independent set is trivial, and therefore it is easy to find **some** stable outcome. However, finding a PNE with a maximal social welfare (SW) (i.e., a PNE with minimal number of players taking the action) is computationally intensive (Jackson & Zenou, Games on networks, 2014). Such PNEs are commonly considered **efficient**. Nevertheless, some outcomes may possess greater SW, but some players may remain unsatisfied in such outcomes. This tradeoff between efficiency and stability motivates the use of an incentive mechanism to promote stability in efficient states.

A possible approach in order to incentivize unsatisfied players to agree on some preferred outcomes, is the **side payments** mechanism. Side payments allow transfers of utility (money) between players, such that players who gain from some outcome may want to pay others that are unsatisfied by it. Side payments can help promote efficiency by providing incentives to some players for seeing more fully the impact of their actions (Jackson & Wilkie, 2005). Regarding the “best-shot” game, one expects that in efficient outcomes “central players” (i.e., players with many neighbors) will take the action, while “peripheral players” (i.e., players with few neighbors) will free-ride their neighbors (Galeotti, Goyal, Jackson, Vega-Redondo, & Yariv, 2010). Intuitively, side payments allow central players to take the action and be compensated by peripheral players.

The current project extends a previous work on the “best-shot” game (Komarovsky, Levit, Grinshpoun, & Meisels, 2015). The “best-shot” game presents different search problems that one may wish to investigate (which are presented in Section 2). Here, some of the previous theoretical results are utilized in order to achieve an improved search process for these problems based on an **evolutionary approach**. Originally, the search process was exhaustive and hence limited only to networks of relatively small size. The proposed genetic algorithm enables to address networks of much greater size within reasonable time (and quality of solutions). The plan of the report is as follows. In Section 2 the game is defined formally and related theoretical results are presented (without their proof). In Section 3 we present the methods and algorithms employed in the current project, while Section 4 shows an overview of the software. Section 5 describes the experimental evaluation and results, and a conclusions are given in Section 6.

2. Problem Description

The current work considers two different search problems defined by the "best-shot" public goods game. In the first search problem, we assume that side payments are allowed and wish to find a strategy that maximizes SW. Since the achieved solution might not be stable, we use an improvement process which ensures the ability to enforce a PNE using side payments. Both improvement process and side payments are computed efficiently (i.e., in polynomial run time). As for the second search problem, we assume that side payments are not allowed and aspire to find an outcome which is both **stable** (PNE) and **maximizes SW** (i.e., there is no other PNE with greater SW). In fact, the first search problem (Max-SW) is equivalent to searching a ***maximal independent set of minimum size*** in an undirected graph, while the second search problem (Best-PNE) is equivalent to searching a ***minimum dominating set*** in an undirected graph. Both problems were proved to be NP-hard (Garey & Johnson, 1979).

2.1. The Game Model

A common and compact representation for network games is the graphical model proposed by Kearns et al. (2001), where a game is based on some underlying graph describing the players' interactions network. Consider a finite set of players $N = \{1, \dots, n\}$ who are connected in a network $G = \{N, E\}$. Each vertex in G represents a player, and edges represent the interaction structure in the game. Given a player $i \in N$, denote the set of i's neighbors by N_i. These are the players whose actions impact i's payoff. The *neighborhood* of i is the set $\{i\} \cup N_i$. Each player chooses an action $a_i \in S_i = \{T, F\}$. Denote the choice $a_i = T$ as *taking the action*, and the choice $a_i = F$ as *avoiding* it. The *utility* of player i for some outcome $v \in S = S_1 \times \dots \times S_n$ and action-cost $c, 0 < c < 1$, is defined as follows:

$$u_i(v) = \begin{cases} 1 - c, & if\ a_i(v) = T \\ 1, & if\ a_i(v) = F \wedge \exists j \in N_i : a_j(v) = T \\ 0, & if\ a_i(v) = F \wedge \forall j \in N_i : a_j(v) = F \end{cases}$$

We say that a player i has a *null-neighborhood* if $a_i(v) = F$, and $\forall j \in N_i : a_j(v) = F$.

Given a "best-shot" public goods game on a network $G = \{N, E\}$, and an outcome $v \in S$, the possible *states* of a player $i \in N$ can be divided into the next four distinct types:

1. TT: $a_i(v) = T\ and\ \exists j \in N_i : a_j(v) = T$

2. TF: $a_i(v) = T \text{ and } \forall j \in N_i: a_j(v) = F$
3. FT: $a_i(v) = F \text{ and } \exists j \in N_i: a_j(v) = T$
4. FF: $a_i(v) = F \text{ and } \forall j \in N_i: a_j(v) = F$

Note that the states TF, FT are **stable** from a player's point of view. A player in one of these states has no incentive to change her strategy unilaterally. The states TT, FF are not stable. A player in state FF (with a null-neighborhood) would prefer to change her strategy to T, and a player in state TT would prefer to change her strategy to F.

2.2. Stability and Efficiency in the "Best-Shot" Game

Nash Equilibrium is an important notion in game theory. We say that an outcome is a PNE, if every player does not prefer to change her own strategy, given the strategies of all other players in this outcome. It is considered desirable to arrive at PNE outcomes that are efficient. One of the most common definitions for efficiency is ***Social Welfare*** (SW) (Osborne & Rubinstein, 1994). The following definitions model the two search problems more formally.

Definition 2.2.1. (PNE)

Given a "best-shot" game G, an outcome $v \in S, v = \langle v_1, \dots, v_n \rangle$, is called a ***PNE*** if $\forall i \in N: u_i(v) \geq u_i(\langle v_1, \dots, v_{i-1}, \neg v_i, v_{i+1}, \dots, v_n \rangle)$. I.e., no player i can achieve better payoff by unilaterally changing her strategy from v_i to $\neg v_i$.

Definition 2.2.2. (SW)

Given a "best-shot" game $G = \{N, E\}$, the ***SW*** of outcome $v \in S$ is defined to be $SW(v) \coloneqq \sum_{i=1}^{n} u_i(v)$.

Definition 2.2.3. (Max-SW)

Given a "best-shot" game $G = \{N, E\}$, an outcome $v \in S$ is a ***Max-SW*** if: $\forall v' \in S: SW(v) \geq SW(v')$.

Definition 2.2.4. (Best-PNE)

Given a "best-shot" game $G = \{N, E\}$, an outcome $v \in S$ is a ***Best-PNE*** if:

a. v is a PNE.
b. $\neg \exists v' \in S: (v' is\ a\ PNE) \land (SW(v') > SW(v))$

Definition 2.2.5. (Side Payments)

Side payments are defined as a transfer function between players, where each player may pay (or receive payment from) each one of its neighbors (Jackson & Wilkie, 2005).

1. Given a "best-shot" game $G = \{N, E\}$, side payments are defined by a transfer function $\tau: N \times N \times S_1 \times \ldots \times S_n \to R^+$, where $\tau_{i,j}(v)$ denotes the payment being transferred from player i to player j in outcome v. We restrict our attention to transfer functions such that if $j \notin N_i$ then $\forall v \in S: \tau_{i,j}(v) = 0$.
2. Given some outcome v in game G
   - The *incoming transfer* of each player i in outcome v is $\tau_i^{in}(v) \coloneqq \sum_{j \in N_i} \tau_{j,i}(v)$
   - The *outbound transfer* of each player i in outcome v is $\tau_i^{out}(v) \coloneqq \sum_{j \in N_i} \tau_{i,j}(v)$
3. We say that some outcome v in G is Side Payments Enforceable (**SPE**) if there exists a transfer function τ defining side payments in G, such that:
$$\forall i \in N \forall v_i' \in S_i: u_i(v) + \tau_i^{in}(v) - \tau_i^{out}(v) \geq u_i(v_i', v_{-i})$$

Proposition 2.2.6. (Best-Response Convergence)

Best-response dynamics in a "best-shot" public goods game converge into a PNE within at most $(2 \cdot n)$ steps. I.e., for every outcome $v \in S$ if we let the players play in **any** order, when in each turn a player reacts with her best-response (unilaterally maximizes her utility, assuming all other players remain with the same strategy), after each player played at most 2 turns a PNE will be reached.

Proposition 2.2.7. (PNE Enforcement Using Side Payments)

1. A Max-SW outcome is SPE.
2. For each outcome $v \in S$, it is possible to find (in polynomial run-time) an outcome $v' \in S$ such that $\forall i \in N: u_i(v') \geq u_i(v)$ **and** v' is SPE.

I. Proofs for 2.2.6 and 2.2.7.1 are in (Komarovsky, Levit, Grinshpoun, & Meisels, 2015).
II. Proposition 2.2.7.2 can be proved in a similar manner to the proof of 2.2.7.1. An intuition will be given in Section 3.2.

The above propositions become very helpful if one wishes to search for the Best-PNE/Max-SW outcome. Once a non-exhaustive algorithm is used, it is possible that only a sub-optimal solution will be found. Hence, some desirable properties of the solution cannot be enforced, such as stability or maximum SW. Proposition 2.2.6 helps to guarantee that some candidate solution will become a PNE, while proposition 2.2.7 ensures that an unstable outcome will be enforced using side payments if necessary. Both improvement processes are explained with more detail in Section 3.

3. Methods and Algorithms

We will present our attempts of solving each of the search problems using an evolutionary approach. Each problem instance was represented as an undirected graph forming the network of players in a “best-shot” game, and for both problems we eventually used the same general model of a ***simple genetic algorithm*** which is presented below.

3.1. Simple Genetic Algorithm for Max-SW and Best-PNE

Given a network and a configuration for the algorithm, do:

```
1.  Population ←initPopulation()
2.  While(Termination condition is not met) do:
    2.1. Parents ← parentSelectionOperator.select(population, fitnessEvaluationOperator)
    2.2. Shuffle parents order
    2.3. Compute next generation: for each consecutive pair of parents apply crossover operator
    2.4. Apply mutation operator for each individual in population
    2.5. Population ← survivorSelectionOperator.select(population, fitnessEvaluationOperator)
3.  Solution ← Choose best candidate solution from population
4.  Solution ← solutionImprovementOperator.improve(solution)
5.  Return solution
```

This model was found effective for both search problems, when the ***appropriate operators*** were used. The operators include **fitness evaluation**, **parent selection**, **crossover**, **mutation**, **survivor selection** and **solution improvement**. For each search problem we will describe the different operators used separately in Sections 3.2 and 3.3. Other algorithm configurations are common to both problems and will be presented in Section 5.

3.2. Maximizing Social Welfare with Side Payments (Max-SW)

Given a network, the problem of maximizing SW with side payments requires a search for a **minimum dominating set** in an undirected graph. Such a set would be the set of players choosing to play “True”, and it maximizes the SW in the “best-shot” game being played on this network. Since side payments are allowed, we do not restrict the solution to be a PNE (before applying the side payments), and we use a simple improvement process to ensure the ability to enforce a PNE.

Candidate Solution Representation

As implied by the use of a simple genetic algorithm, each candidate solution is a vector of length n (which is the number of player/vertices), and the $i'th$ coordinate (“allele”) in the vector is assigned with a Boolean value, which represents the action of the $i'th$ player. Hence, each candidate solution is trivially decoded to an outcome in the game (just choose for each player an action according to the appropriate allele).

Fitness Evaluation

Each candidate solution was evaluated according to the SW of the corresponding outcome.

Parent Selection

Two standard parent selection operators were evaluated:

1. **Roulette wheel** (fitness proportional) parent selector – assign each individual a part of a roulette wheel (proportionally to its fitness), and spin the wheel m times to select m individuals.
2. **Tournament** parent selector – for m times choose randomly k individuals from the population, and between them choose the individual with the best fitness.

Crossover

Three standard crossover operators were implemented: **1-point crossover**, **N-point crossover** and **Uniform crossover**. In all crossover operators, a crossover was applied only with some certain probability p_c, and otherwise both parents were copied to the next generation. Since we did not observe major differences in the results of the different operators, we eventually used only the **uniform crossover** operator in our final experiments.

Mutation

We used a simple mutation operator which alters the value assigned for each player with some small probability p_m^1. Mutation was applied only with some certain probability p_m^2.

Survivor Selection

We used an **elitism** survivor selector. Since selective pressure (towards greater fitness) was already achieved by a parent selection operator, we only wanted to preserve the **anytime behavior** of a simple genetic algorithm by using the mechanism of elitism. In order to do so, we chose the top k individuals (with best fitness, from both **parents'** population and **descendants'** population), where k was a predefined constant.

Solution Improvement

After several generations, the simple genetic algorithm return the best candidate solution according to our fitness function, which maximizes SW. Since the returned solution is not necessarily the optimal solution, proposition 2.2.7.1 is not sufficient in order to enforce a PNE using side payments. It is indeed possible that it is not possible to enforce one in the returned outcome. Assume that the algorithm returned some sub-optimal outcome $v \in S$. According to proposition 2.2.7.2 it is would be possible to find an outcome $v' \in S$ such that $\forall i \in N: u_i(v') \geq u_i(v)$ **and** v' is SPE.

We defined a simple solution improver (Max-SW Improver) which finds an outcome v' with the appropriate properties with polynomial run-time. The solution improver only iterates once over all vertices in the given network, and work as follows:

For each vertex v do:

1. If in the current solution v is in state FF (as defined in Section 2.1), simply change its assigned value to T and update the current solution.
2. If in the current solution v is in state TT:
    a. If any of the neighbors of v depend on it (we say that a neighbor of v depends on it if this neighbor chooses F and v is its only neighbor that chooses T), keep the assignment of T for v.
    b. Otherwise, alter v's choice to F.

We claim that the above solution improver guarantees that the resulted outcome v' is SPE. After the improvement process, it is not possible that a player is in state FF. If a player is in state FT or TF she does not wish to deviate unilaterally. Hence, we only need to show it is possible to compensate each player in state TT using side payments. However, since a player stays in state TT only if at least one player depends on her, this player (or group of players) can ensure her stability by transferring a total amount of c using side payments. (This is only an explanation for the **intuition** behind the improvement process, we assume that presenting the formal proof is beyond the scope of the current report).

3.3. Maximizing Social Welfare without Side Payments (Best-PNE)

Given a network, the problem of searching for the best PNE without side payments is very different than finding a Max-SW outcome. In an evolutionary-computation perspective, the main difference is that in the **Max-SW** we aspire to maximize a single-objective fitness function, since the only goal is to maximize SW, and each candidate solution would be ***valid***, while in the **Best-PNE** problem, we wish to compare the SW of only PNE outcomes. Hence, several questions remain, such as: which genetic representation model is the most suitable? And which fitness model would be the most effective?

Candidate Solution Representation and Fitness Evaluation

Finding the suitable representation and fitness models for the Best-PNE problem was probably the most challenging part in our project. We will describe three different attempts, of which only the third (and final) attempt was found successful in our experimental evaluation.

1. Simple representation + multi-objective fitness function **(Best-PNE)** – our first attempt was to keep the same representation model as in the first search problem, and only changing the fitness model such that stable (PNE) outcomes would be preferred over others. In order to do so, the fitness function was defined as follows:

$$fitness(v) = SW(V) + \sum_{i\in N} n \cdot (isStable(v,i))$$

   I.e., the fitness of an outcome is the SW plus a "bonus" of n for each player i which is stable in outcome v. Such a fitness function highly prefers PNE outcomes over others. The preliminary experimental results for this attempt were found not very successful, even when we tried to give different weights for the stability/SW components in the fitness function.

2. Permutation representation + simple fitness function **(Permutation)** – In attempt to find a better model for the Best-PNE problem, we thought it could be beneficial if we limit the search space only to **valid solutions**, meaning that we would only search outcomes that are already PNEs. Among such outcomes, the fitness function can once again become one-dimensional and take into consideration only the SW (assuming that the outcome is a PNE in first place). This goal required a different representation model. We defined each candidate solution once again as a vector of length n, only this time the $i'th$ coordinate in the vector was assigned with an integer value, which represented the **order** of the $i'th$ player. I.e., each individual was a **permutation** of the vertices in the network. In order to decode a candidate solution to an outcome, we performed a best-response process, where the initial outcome is $\langle F, F, \dots, F\rangle$, and according to the given order each player chooses whether to keep her F assignment or to alter it to T. Such decoding process is linear (one pass over all vertices, each update is done in O(Number of neighbors)). It also ensures that every candidate solution is mapped into a single PNE outcome, and that each PNE can be represented by at least one candidate solution. This representation required to implement different mutation and crossover operators which are appropriate for such a

representation. We used all permutations-appropriate mutation operators presented in class: “Swap”, “Insert”, “Inversion” and “Scramble”. For crossover we used order-1 crossover. The parent and survivor selection operators remained the same as before, and the fitness evaluation was again done only according to the SW of a candidate solution. This attempt also failed to return results with reasonable quality. We assume that the main reason is that although we limited the search space only to PNE outcomes, we significantly increased the search space, since now many genotypes can be mapped to the same phenotype. Combinatorically, the first representation included 2^n possible genotypes, and the second representation included $n!$ possible genotypes.

3. Simple representation + special fitness function **(Best-potential PNE)** – Our final attempt was representing a candidate solution again in a simple manner (each allele is a Boolean value), and using a different fitness function. In order to avoid the multi-objectiveness, we decided to decode each candidate solution to a PNE, before evaluating its fitness. In this manner, the fitness function is once again only based on a one dimension – the SW, and not on stability. We defined a fitness evaluator which first improve the given outcome to a PNE (using a best response process done according to some predefined, constant, order on the players). Then, the SW of the resulted PNE is the fitness of the original candidate solution. Once a final solution is found, we use the same decoding process as the solution improver, so the fitness of each candidate is in fact also the SW of a solution based on this candidate. For mutation and crossover we used the same operators as in the first search problem (Max-SW), as well as the different selection operators. This last attempt was found to be the most successful in our experimental evaluation, and was used in the final version of our project, for solving the Best-PNE problem.

3.4. Problem Generation

In order to evaluate the proposed algorithms, a problem-generation method was required. For that purpose, we used **Barabási-Albert scale-free networks** (Barabási & Albert, 1999). Barabási-Albert scale-free networks are defined by 4 parameters: n, m_0, ep and m, where: n is the total number of vertices, parameters $m_0 \leq n$ and $ep \in [0,1]$ define a basic random network (m_0 is the number of vertices in this basic network and ep is the edge probability for every pair of vertices). After a random basic network is established, we add new vertices to the network until the graph consists n vertices. The parameter $m \leq m_0$ defines the number of links created from each new vertex to an existing one. The probability for an existing vertex to be chosen as a neighbor to a new vertex is proportional to its degree. In that manner, the graph created typically consists a **few** “hubs” – central players with many neighbors, and **many** peripheral players with only few neighbors. The values for the different parameters we used in our experimental evaluation are detailed in Section 5.

4. Software Overview

In this section we give an overview of the implementation of the proposed algorithms and the experimental evaluation. Our (**Java**) code is completely **open-source**, so all **source files**, as well as the current **report** and **theoretical results** can be found at:
https://github.com/zoharkom/GeneticAlgorithmPGG

4.1. Problem Generation and Representation

The representation and generation of problems are done by the "**graph"** package. Its main components are as follows:

- **Graph** – this class represents an undirected graph instance, enables to add/remove vertices and edges, and to receive some basic information regarding a graph instance (for example, getting the neighbors of some vertex).
- **ScaleFreeGraphGenerator** – this class enables to generate a random instance of a Barabási-Albert scale-free network according to the given parameters (as described in Section 3.4).
- **GraphUtils** – this class proposes an API which supports all graph-related computations needed for our algorithms. Sample procedures are:
    - **a.** **isTT/isTF/isFT/isFF –** given a CandidateSolution and a Graph and a Vertex in the given graph, this methods return true if and only if the given vertex is in state TT/TF/FT/FF under the given assignment (i.e., CandidateSolution) for this graph.
    - **b.** **hasDependents** – given a CandidateSolution and a Graph and a Vertex in the given graph, this methods return true if and only if the given vertex has at least one neighbor which depends on it (as defined in Section 3.2) under the given assignment.
    - **c.** **GDFWriter** – given a Graph and a Candidate solution, the GDFWriter enables to write the graph with the given assignment in "gdf" format, which enables to view the result using a graph visualization software (we used "Gephi").

4.2. Solution Representation

The "**algorithm.components"** package include two important classes:

- **CandidateSolution** – an instance of this class represents an individual in the evolutionary algorithms. In addition to several constructors it supports **getAllele**/**setAllele** methods and **equals**.
- **Population** – simply a container with a fixed-size number of CandidateSolutions.

4.3. Algorithm Overview

The **"algorithm"** package contains all algorithms implemented in order to solve the "best-shot" public goods game. We implemented one general evolutionary algorithm called "SimpleGeneticAlgorithm", and two non-evolutionary algorithms called "BruteForceAlgorithm" and "BestResponseAlgorithm". All algorithms implement the "PGGAlgorithm" interface, which require the implementation of a single method: `public CandidateSolution findSolution(Graph g, ArrayList<double[]> stats)`.
Given a graph g, this method return a CandidateSolution (guaranteed to be optimal only in the "BruteForceAlgorithm"), and also fills some statistics into the given "stats" parameter.

- **SimpleGeneticAlgorithm** – This is the general evolutionary algorithm, implemented according to the logic described in Section 3.1. This algorithm was used for both search problems by generating different configurations for the different problems. The configurations determine which operators will be used and will be described in detail later.
- **BruteForceAlgorithm** – Returns an optimal solution according to some fitness function. We simply went over all possible solutions. For each one the fitness was evaluated, and the one with the best fitness was chosen as the optimal.

- **BestResponseAlgorithm** – Chooses a random initial assignment to each vertex, than let the players play with best response dynamic according to some order. According to Proposition 2.2.6 this method ensure convergence to some PNE outcome within linear run-time (linear w.r.t. graph size).

4.4. Genetic Algorithm Configuration

The "**algorithm.config**" package contains a configurations class for each search algorithm. In fact, the best response algorithm does not require any configurations, and the brute-force algorithm only requires to define a fitnessEvaluator. The configuration for the simple genetic algorithm is more extensive and requires to define all operators described in the next subsection, as well as the number of generations of the algorithm, probabilities for mutation and crossover, a seed for the random number generator (so a "random" search process could be repeated if necessary and so on. This implementation was generic enough to enable the use of the same algorithm for different problems, or different search processes for the same problem.

4.5. Operators

The "**algorithm.operators**" package contains the following sub-packages: "**crossover", "mutation", "fitness", "selection",** and **"improve"** (the last package is for the solution improvement operator). The "crossover"/"mutation"/"fitness"/"improve" packages all contain **one** interface to describe the operation, and several implementations according to the logic described in Section 3. The "selection" package contains **two** interfaces – one for **parent selection** and another for **survivor selection**, and again – several implementation for each according to the described logic. Due to space limitation we avoid a complete description of the operators which are rather self-explanatory (and can be viewed at the project's repository).

4.6. Simulation

Finally, the "**simulation**" package includes the simulator used for the experimental evaluation. This class contain the main procedure of the program, which configure the algorithms and problems tested in an experiment. The simulator creates a certain number of problem instances (graphs), for each instance we find a solution with each of the algorithms (which were previously configured), and collect the statistics from each search process. The graphs and solutions are written in "gdf" format, and the statistics in a "csv" format. The output files were used to examine the results of the experiments visually using "Gephi" and MS "Excel".

5. Experimental Results

An experimental evaluation was required in order to examine the properties of the proposed genetic algorithms. **Run time** and **quality of solutions** of the proposed genetic algorithms were compared in different settings for both the Max-SW problems and the Best-PNE problems. For each problem type, we have conducted experiments with 2 different settings, one for small networks and another for large networks. This differentiation is due to the fact that in small networks it is possible to compute (with reasonable time) the optimal solution, and compare it with the results of the proposed genetic algorithms. Even though no complete algorithm could compute in reasonable time the results for larger networks, it was interesting to examine the behavior of the genetic algorithms in comparison with the best-response algorithm on such networks. Overall, 4 experiments were conducted.

The common settings in all experiments:

- Action cost (i.e., player's cost for choosing T): 0.4
- Mutation probability (1): 1 (probability of an offspring to go through mutation)
- Mutation probability (2): 0.1 (the probability for an allele to change, given that mutation occurs)
- Crossover probability: 0.6
- Population size: 100
- Elite size: 20
- Tournament size (for the tournament parent selector): 5

The settings common for **small** networks:

- Graph generation parameters: n = {10, 12, 14, 16, 18, 20}, m_0=4, $ep = 1, m = 1$
- Number of generations: 20

The settings for **large** graphs networks:

- Graph generation parameters: n = {50, 100, 150, 200, 250, 300}, m_0=10, $ep = 0.75, m = 1$
- Number of generations: 200

For each setting 100 different networks were generated, and summed to a total of $(4\ settings) \cdot (6\ graph\ sizes) \cdot (100\ instances) = 2400$ different networks. Each such network was solved by 3 to 4 different algorithms: Brute force, best response, GA1, GA2. (For large networks we could not use the brute force algorithm, and the operators for GA1 and GA2 are described in each experiment separately). A summary of the results is described below.

5.1. Maximizing Social Welfare with Side Payments (Max-SW)

In these experiments our aim was to find a solution to the Max-SW problem as defined in Section 2. Such solutions are not necessarily PNEs, however by using Proposition 2.2.7.2 and the solution improvement process described in Section 3.2, a PNE can be enforced in (low) polynomial time using side payments (and the same SW remains after applying this algorithm).

Specific settings:

GA1 operators: Tournament parent selector, simple mutation operator, uniform crossover operator, elitism survivor selector, Max-SW fitness evaluator, Max-SW solution improver.

GA2 operators: Roulette-wheel parent selector, all other operators are the same as in GA1.

Small networks

Figure 1 shows the average quality of solutions of the different algorithms for small Max-SW problems:

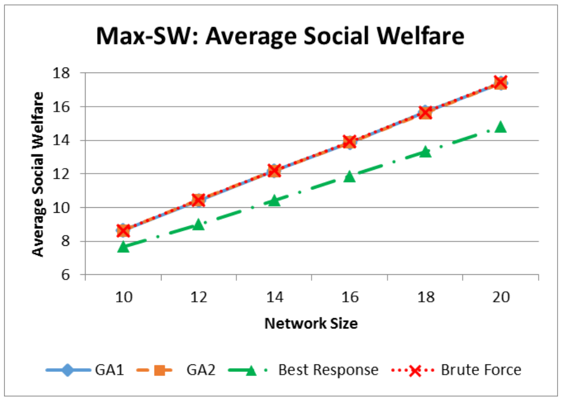

Figure 1: Average SW for Max-SW on small networks

Optimal Solution Rate		
	GA1	**GA2**
10	100%	99%
12	99%	100%
14	95%	94%
16	90%	87%
18	96%	91%
20	89%	84%

Table 2: Optimal solutions rate for Max-SW

We can observe that the solutions returned by both of the genetic algorithms were almost always optimal, and the average SW in those solutions was almost the same as the average SW of the optimal solution. Even in cases that the retuned solution was sub-optimal, its SW was only slightly worse than the optimal SW. The optimal solution rate, as described in Table 1, implies that GA1 return better solutions than GA2, since (except for the case of 12 players) the success rate of GA1 is always greater. We can also observe that the optimal solutions are much better than those returned by a simple best-response search.

Figure 2 shows the average run time required for the different on the same networks. Note that the run time was measured in milliseconds, and it is displayed on a **logarithmic scale**. We can observe significantly

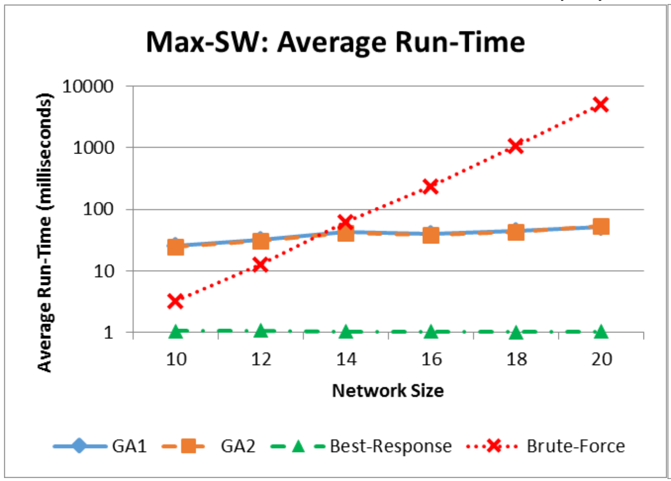

Figure 2: Average run-time for Max-SW on small networks

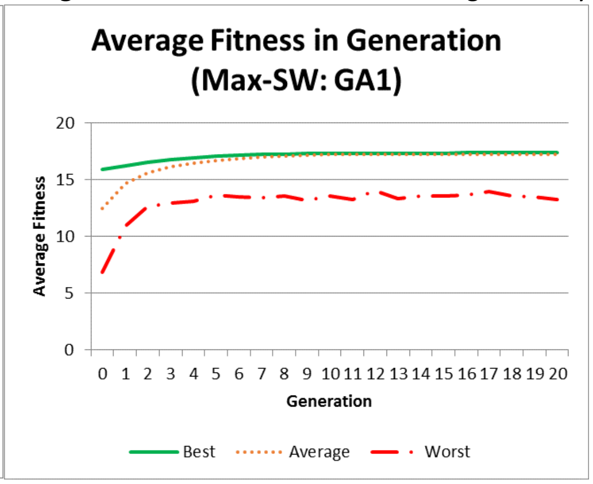

Figure 3: Average fitness over generations for Max-SW on networks with 20 vertices

lower increase in run-time for both genetic algorithms in comparison to the brute-force algorithm as the size of the network increase. However, it is important to note that as the size of the network increase, more generations are required in order to obtain optimal (or at least high-quality) results. The average improvement of individuals in the population through generations is demonstrated in Figure 3. The graph shows the average of: the best fitness of an individual in each generation, the worst one, and the average over all the individuals in the population of GA1 when solving networks with 20 vertices.

It can be seen that the algorithm was able to find the best solution rather quickly. In fact, most of the population improved its fitness as the average fitness is very close to the best individual, while the population is still diverse as the worst solution, usually, has a considerably lower fitness compared to the average. Although GA1 converged around 10 generations, Table 1 suggests that if more generations were allowed, the success rate could be improved. Since the proposed algorithms are **anytime algorithms**, the tradeoff between run-time and quality of results can be parameterized according to the preferences of the user.

Large networks

After we were convinced in the effectiveness of the algorithm for smaller networks, we wanted to see its behavior on large networks of 50 to 300 players. Unfortunately there was no way to perform a brute-force search on those graphs that would terminate within reasonable time. Hence, the actual value of the optimal individual is remained unknown.

Consider the following results:

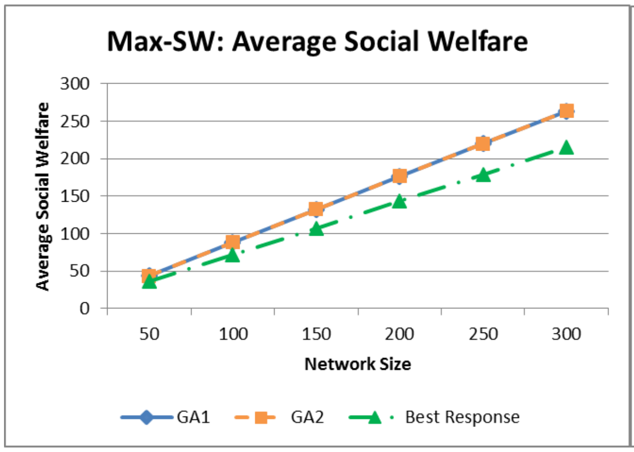

Figure 4: Average SW for Max-SW on large networks

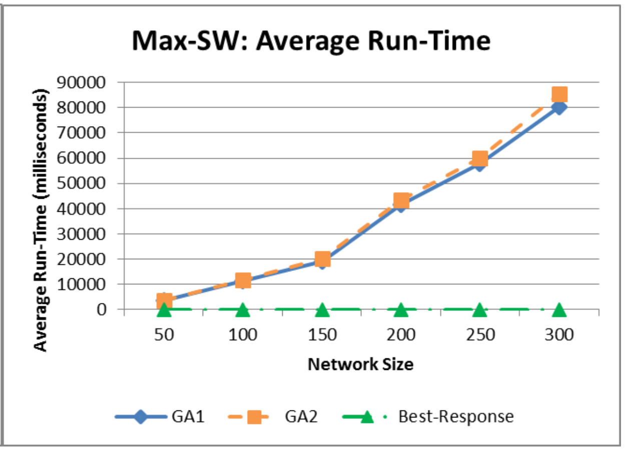

Figure 5: Average run-time for Max-SW on large networks

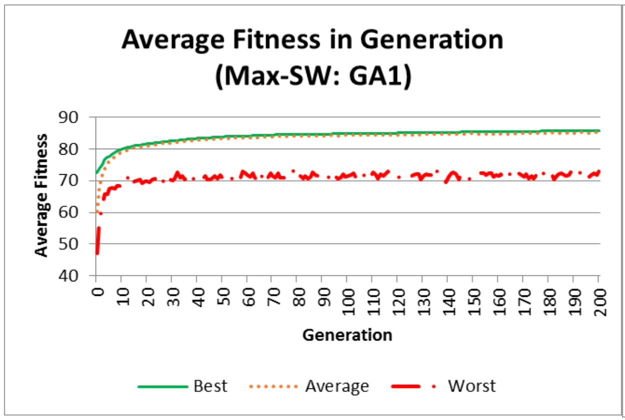

Figure 6: Average fitness over generations for Max-SW on networks with 100 vertices

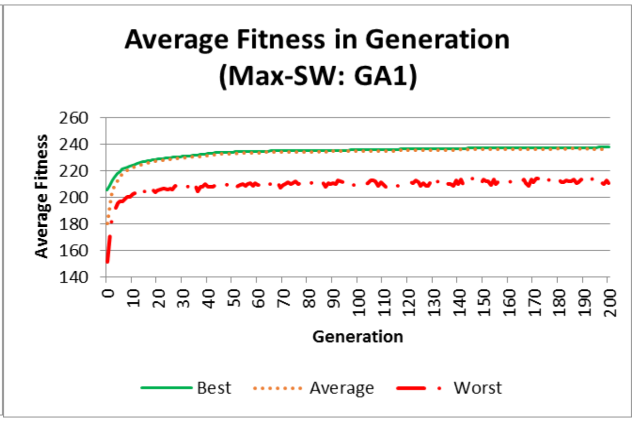

Figure 7: Average fitness over generations for Max-SW on networks with 300 vertices

As expected, the best individual was also achieved in a relatively early generation. As the number of the players grows – the problem becomes harder, therefore it takes more generations for the genetic algorithms to converge. Recall that for 20 players it took around 10 generations, for 100 players it took roughly 140 generations and for 300 players 190 generations – so the algorithm scales well with respect to the quality of solutions. In addition, Figure 5 demonstrates scalability w.r.t the run-time required.

5.2. Maximizing Social Welfare without Side Payments (Best-PNE)

Here we address the Best-PNE problem, again for small and large networks separately.

Specific settings:

GA1 operators: Tournament parent selector, simple mutation operator, uniform crossover operator, elitism survivor selector, best-potential-PNE fitness evaluator (described in Section 3 as attempt #3), best-response solution improver.

GA2 operators: Best-PNE fitness evaluator (described in Section 3 as attempt #1), all other operators are the same as in GA1.

Small networks

The results for Best-PNE problems on small networks are as follows:

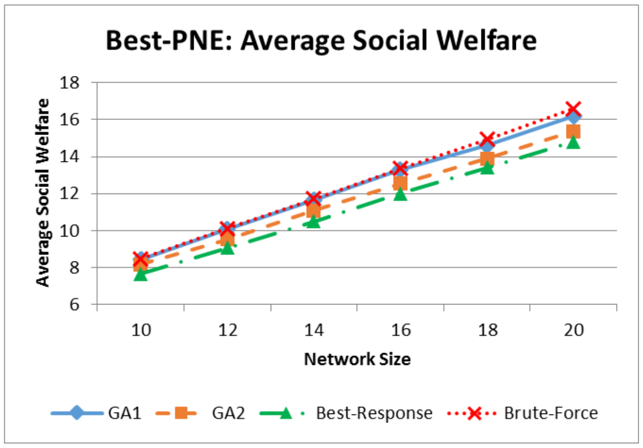

Figure 8: Average SW for Best-PNE on small networks

Optimal Solution Rate		
	GA1	**GA2**
10	99%	52%
12	96%	36%
14	88%	29%
16	94%	25%
18	75%	19%
20	73%	14%

Table 2: Optimal solutions rate for Best-PNE

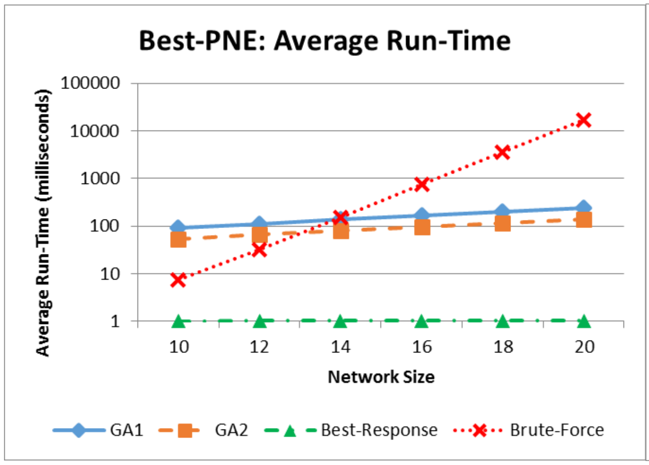

Figure 9: Average run-time for Best-PNE on small networks

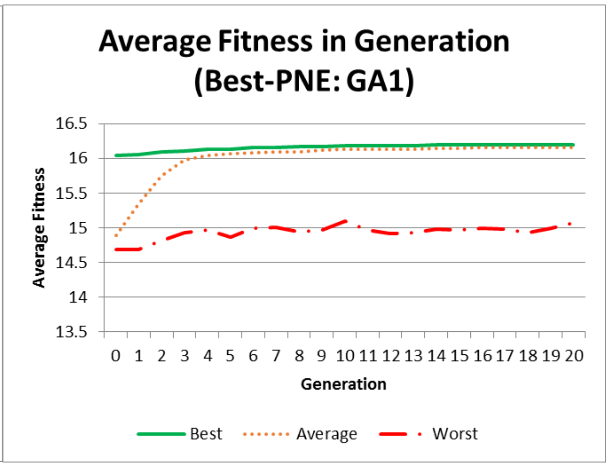

Figure 10: Average fitness over generations for Best-PNE on networks with 20 vertices

Figure 8 shows that the average quality of solutions is again almost perfect for GA1. Table 2 demonstrates the superiority of GA1 over GA2. For both algorithms we see lower success rate in comparison to the case of Max-SW. However, GA1 still very likely to return an optimal solution and again – when sub-optimal solutions were returned, their quality was just slightly lower than optimal. This is of course not the case when Best-PNE fitness evaluator was used (GA2). As we can see GA2 is not able to find the best solution in most cases even on a very small networks. We can also see that in some of the cases the solution found is very close to the one returned by the best-response algorithm, which runs much faster (less than 1 millisecond) and is much easier to implement. Run-time results, and quality through generations are similar to the case of Max-SW.

Large networks

The results for Best-PNE problems on large networks are as follows:

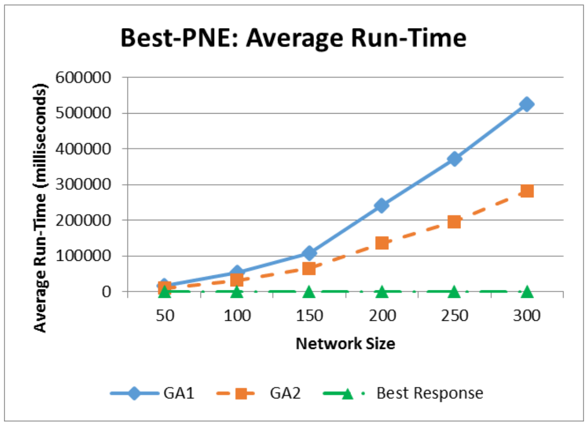

Figure 11: Average run-time for Best-PNE on large networks

Best-PNE: Average Social Welfare			
	GA1	GA2	Best Response
50	38.89	37.82	36.24
100	75.66	72.44	71.49
150	114.04	108.60	107.72
200	151.68	144.00	142.96
250	189.56	181.12	180.04
300	226.76	216.84	215.68

Table 3: Average SW for Best-PNE on large networks

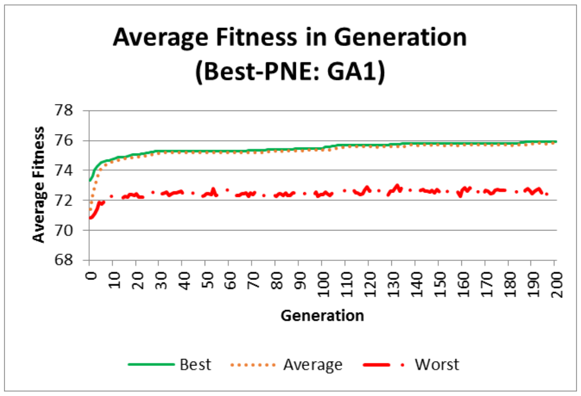

Figure 12: Average fitness over generations for Best-PNE on networks with 100 vertices

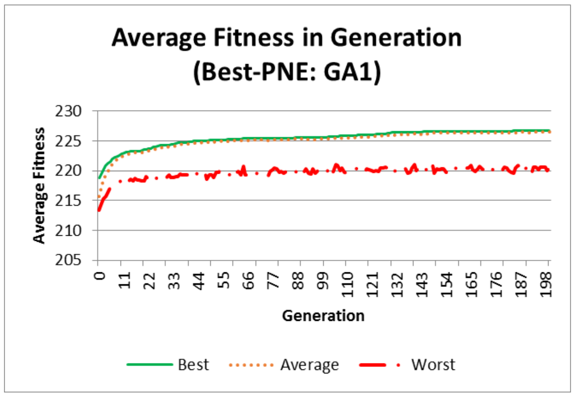

Figure 13: Average fitness over generations for Best-PNE on networks with 300 vertices

Overall, the results are aligned with the previous observations. We can see that GA1 requires more time than GA2 and of course both require much more time than the simple best-response algorithm. The same ratio between these 3 algorithms remains also when the quality of results is considered. I.e, GA1 produces considerably better solutions than GA2, and GA2 produces better solutions than best-response. Regarding the improvement through generations we can observe lower increase pace, and as the size of the network increase we can see that it takes more generations for the algorithm to converge.

6. Conclusions

An evolutionary approach for towards the “best-shot” public goods game was investigated. Two simple genetic algorithms were proposed and examined for the problems of maximizing SW in an equilibrium with and without side payments. The combination of a genetic algorithm and a local improvement process was found very effective for both the Max-SW problem and the Best-PNE problem. This combination allowed to achieve **valid** results, with optimal (or almost optimal) **quality** within reasonable **time**. Moreover, the use of **elitism** brought to an **anytime** behavior of the genetic algorithms. Hence, one would be able to determine the wanted number of generations in order to achieve certain quality/run-time according to her preferences regarding this tradeoff.

6.1. Maximizing Social Welfare with Side Payments (Max-SW)

Tournament vs. Roulette-Wheel Parent Selection

We found the tournament parent selection to perform better than roulette-wheel parent selection for Max-SW problems. We explain this phenomenon by the fact that two outcomes may possess almost the same SW, however if one of them is even slightly better – we would like the evolutionary process to prefer it **significantly** over the other. In roulette-wheel selection we also prefer the outcome with greater SW, but the parent selection procedure is very sensitive to the number of players and the cost of taking the action. The tournament parent selection overcomes this difficulty in our case.

Effectiveness of the Evolutionary Approach

The basic evolutionary functions (selection, fitness, mutation, etc.) were enough in order to reach very good solutions - the best possible for graphs with 20 vertices, and for graphs with 50 to 300 vertices we were able to converge into solutions which we assume are the best possible or very close to is, however, testing this theory is not possible due to run-time limitations. We can see that in this case we get a 'stable' solution (the best is not improved) pretty early in the run, however, in the second problem (Best PNE) it takes more time for the best solution to stabilize. In both of the cases there is a very large improvement in the early generations, and later it becomes more challenging to improve the current-best so there are minor improvements along the run, until stabilization. We can also observe that in both problems the majority of the population is concentrated around the best solution and that there is a big gap between the quality of the best solution and the worst - a fact which by crossover allows to improve and get fitter individuals.

6.2. Maximizing Social Welfare without Side Payments (Best-PNE)

This problem proved itself harder than the previous one, and finding a fitting representation and fitness function took a few failed attempts.

Multi-objective vs. Single-objective Fitness Function

We first experimented with the "Best-PNE" fitness evaluator, which defined a multi-objective fitness function that considered both **stability** and **SW.** This fitness function did not achieve the results we hoped for, and was considerably far from the best solution achieved by the "brute-force" algorithm. This was due to the fact that the fitness evaluation took into account the current state of the solution, which is more often than not - not a PNE. Since we are only interested in legal solutions (PNEs), we had to use a post-processing mechanism in order to convert the solution into a PNE (the “best-response” solution improver). Using the “Best-PNE” fitness evaluator, the stability of the solution became more dominant than its SW. On the other hand, when more weight was given to SW, we got invalid solutions, and after using the solution improver, their SW has decreased dramatically. When moving to the “Best-potential-PNE” fitness evaluator, the calculation of the fitness is not only based on the current state of the solution, but also on the quality of the solution after converting it into a PNE. This way we know which candidate solutions are actually promising, and which are misleading.

Representation Influence on Performance

In attempt to apply a representation model that will allow only valid solutions, we also tried to represent a candidate solution as a permutation (as explained in Section 3.3). A preliminary evaluation process showed terrible results. We explain the bad results by the fact that although this representation modal enabled to move from multi-objective to single-objective fitness function, its effect on the size of the search space was tremendous. We went from 2^n possible candidate solutions to $n!$. This change is probably the reason that this representation model was not found effective.

Effectiveness of the Evolutionary Approach

After the “right” model for representation and fitness evaluation was found, we reached much better results. On small networks - which contain 10 to 20 vertices, the algorithm almost always achieved the optimal solution. Moreover, the best individual was found fairly quickly - in the first few generations, so here we can see that an optimal result can be achieved very quickly by using our evolutionary algorithm, instead of running the "brute-force" algorithm which requires much more computational effort.

In conclusion, the proposed algorithms were found scalable and effective for the defined search problems, both in perspective of run-time and quality of results.

7. Rreferences

Barabási, A. L., & Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286(5439), 509-512.

Bramoullé, Y., & Kranton, R. (2007). Public goods in networks. *Journal of Economic Theory*, 135(1), 478-494.

Galeotti, A., Goyal, S., Jackson, M. O., Vega-Redondo, F., & Yariv, L. (2010). Network games. *The review of economic studies*, 77(1), 218-244.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: a guide to the theory of NP-completeness.* San Francisco, LA: Freeman.

Hirshleifer, J. (1983). From weakest-link to best-shot: The voluntary provision of public goods. *Public choice*, 371-386.

Jackson, M. O., & Wilkie, S. (2005). Endogenous games and mechanisms: Side payments among players. *The Review of Economic Studies*, 72(2), 543-566.

Jackson, M. O., & Zenou, Y. (2014). Games on networks. *Handbook of game theory*, 4.

Kearns, M., Littman, M. L., & Singh, S. (2001). Graphical models for game theory. *In Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pp. 253-260.

Komarovsky, Z., Levit, V., Grinshpoun, T., & Meisels, A. (2015). Efficient Equilibria in a Public Goods Game. *The 2015 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-2015), Singapore*, **(under review)**.

Osborne, M. J., & Rubinstein, A. (1994). *A course in game theory.* MIT press.