A Genetic Algorithm Approach to Compute Mixed Strategy Solutions for General Stackelberg Games

A Genetic Algorithm Approach to Compute Mixed Strategy Solutions for General Stackelberg Games

Srivathsa Gottipati
International Institute of Information Technology
Hyderabad, India
srivathsa.gottipati@research.iiit.ac.in

Praveen Paruchuri
International Institute of Information Technology
Hyderabad, India
praveen.p@iiit.ac.in

***Abstract*—Stackelberg games have found a role in a number of applications including modeling market competition, identifying traffic equilibrium, developing practical security applications and many others. While a number of solution approaches have been developed for these games in a variety of contexts that use mathematical optimization, analytical analysis or heuristic based solutions, literature has been quite sparse on the usage of Genetic Algorithm (GA) based techniques. In this paper, we develop a GA based solution to compute high quality mixed strategy solution for the leader to commit to in a General Stackelberg Game (GSG) using a normal form game formulation. The leader faces multiple types of followers with discrete utility functions where the mixed strategy of the leader (but not the actual action taken in the round) is known to the follower. Our experiments showcase that the GA developed here performs well in terms of scalability and provides reasonably good solution quality in terms of the average reward obtained. Given that finding the optimal mixed strategy solution for GSGs is NP-hard (and the optimal solution for leader lies in the mixed strategy space), we believe that the solution approach presented here can support further development of practical applications using GSGs.**

***Index Terms*—Genetic Algorithms, General Stackelberg Games, mixed strategy**

I. Introduction

Stackelberg games have found a role in a number of applications including modeling market competition [1], identifying traffic equilibrium [2], practical security applications [3]–[6] and many others. General Stackelberg Games (GSGs) [7]–[9], assume a player referred to as the leader, who can commit to a strategy to optimize its utility function while other players referred to as followers respond to the leader's decision to optimize their own utility functions. GSG's can be expressed as bilevel optimization problems, where the top level represents the leader's decision problem while the followers' responses are included as the optimal solution to the second level problem [10], [11].

When the leader in a GSG faces a single follower, the problem can be solved in polynomial time [12]. [12] also shows that if there are multiple followers, the problem is NP-hard. Hence, there is a need to develop algorithms that can scale better. Although a Genetic Algorithm (GA) based solution may not guarantee optimal solution, with good design it is possible to generate high quality solutions. To showcase the advantage of being a leader in GSG, we borrow the following example from [13]. Consider a Stackelberg game with the payoff table expressed as a normal form game [12], as shown in Table I. The leader is the row player, follower is the column player and both the players move simultaneously (or do not observe the other player's move before making their own).

	c	d
a	2,1	4,0
b	1,0	3,2

TABLE I: Payoff table for an example GSG

A pure-strategy Nash equilibrium [7] for the game in Table I, is when the leader plays a and the follower plays c which gives the leader a payoff of 2. There can be multiple such pure Nash equilibrium strategies in general, but there is only one such for this game. However, in the Stackelberg version of the game, if the leader can commit to playing b, the leader will obtain a payoff of 3, since the follower would play d to ensure a higher payoff for itself. Let's assume that the leader commits to a uniform mixed strategy of playing a and b with equal (0.5) probability, then the follower will play d, leading to a payoff of 3.5 for leader. The optimal mixed (Stackelberg) strategy for this game, would result in a leader strategy [a,b] = [2/3, 1/3] and follower action b, leading to a payoff of 3.67 for the leader. As shown in literature, the optimal strategy for the leader in a Stackelberg game would in general be a mixed strategy [12], hence most works in this space focus on identifying high quality or optimal mixed strategy solutions in an efficient manner.

A. General Stackelberg Games (GSGs)

A GSG contains N agents, and each agent n must be one of a given set of types θ_n. The GSG we consider has two agents namely a leader and a follower. θ_1 is the set of possible types for the leader and θ_2 is the set of possible types for the follower. We consider similar scenario as used in [13], where a leader (security agent of one type) would face multiple follower types (i.e., robber). During the game, the robber knows its type but the security agent does not know the robber's type. For each agent (security agent or robber) n, there is a set of strategies σ_n and a utility function $u_n : \theta_1 \times \theta_2 \times \sigma_1 \times \sigma_2 \rightarrow \Re$.

978-1-7281-8393-0/21/$31.00 ©2021 IEEE

B. Genetic Algorithm (GA)

As introduced in [14], a Genetic algorithm (GA) is a meta-heuristic commonly used to generate solutions for optimization and search problems by relying on bio-inspired operators such as selection, crossover and mutation. Prior work that focuses on developing GA based solution(s) for Stackelberg game is quite sparse. [15]–[17] aim to develop solutions using GA for a two player Stackelberg game where the utility function of each player is modeled using a continuous mathematical function (and also differentiable in [15]). In contrast, we develop a GA solution to search for high quality mixed strategy for the leader to use in a two player GSG (uses the notion of types for each player) modeled as a normal form game with discrete utilities. We experimentally show that our solution is scalable to solve much bigger games. While games with continuous utility functions are studied in game theoretic literature, a large part of the literature uses the normal form or extensive form game representation where the utility of each player for each possible outcome of the game is typically specified as a real number. Hence the work presented in this paper captures a wide variety of settings and would be of general interest.

II. Genetic Algorithm approach

We now provide specifics of the GA approach we develop for our purposes. We build our fitness function upon the idea presented in [15], which describes computation of Stackelberg solutions using GA in which utilities for the players are modeled as continuous and differentiable mathematical functions. We therefore need to make suitable modifications to account for the discrete utility function we model here. [15] assumes real valued domain as input and output for the leader and the follower, while for our problem we consider integer valued domain as input and output for the follower, and real valued domain for the leader. The selection operation for our purposes is adapted from [18] and crossover from [19]. We develop a custom mutation operation, which is a combination of exhaustive search along with a relaxed version of the same. Custom mutation operations have been used in literature in different contexts e.g., [20] solves a scheduling problem using customized crossover and mutation operations.

A. Population

The initial population for GA plays a key role in achieving optimal solution as it represents the search space for the GA. Hence, we need to provide a suitable initial population to achieve high quality solutions. We use the following seeding technique for our purposes: The initial population is comprised of randomly generated mixed strategies and is seeded with the best deterministic strategy. To compute the best deterministic strategy of the leader, we first compute the reward the leader can obtain for each of its actions. This computation needs the leader to identify the (best) response the follower would choose for that particular action of the leader. We then choose the action which provides highest reward for the leader as the best deterministic strategy. When there are multiple follower types, each follower type would pick a different best response and the follower best response would be a weighted combination of all these best responses (where weight is the type probability). Algorithm 2 summarizes the set of steps involved. Each mixed strategy is generated as follows: We randomly pick the actions (repetition not allowed) and assign a random probability e.g., p_1 between 0 to 1 for the initial action picked. For the next action picked, the range of choosing the probability p_2 is updated i.e., between 0 to $(1 - p_1)$ and this process continues. Each time a probability p_i is assigned to an action, the $1 - sum(p_i)$ is updated until it becomes zero. Pseudo code presented in Algorithm 1 shows the generation of a chromosome in the population. This generation procedure satisfies the following constraints: (a) Probability assigned to each action must be between 0 and 1. (b) Sum of probabilities across all the actions must sum to 1.

Algorithm 1 Population initialisation

```
r = 1
ind is the chromosome of length equal to no.of actions of leader
all values in chromosome are zero initially
for i= 0 to Number of actions of the leader do
    n = pick random action
    if ind[n] == 0  then
    ▷ checking the index of chromosome ensuring same index/action not picked again
        ind[n] = random(0,r)          ▷ assigning random probability between 0 to r
    end if
    r = 1-sum(ind)   ▷ sum(ind) sum of probabilities in ind
    if  sum(ind)==1 or r==0 then
        break
    end if
end for
return ind    ▷ created chromosome included in population
```

Algorithm 2 Deterministic strategy in multi-follower scenario

```
Consider the leader has n actions
let there be m follower types
let w be the vector that contains m weights
for i= 0 to n actions of the leader do
    Compute the total reward for the i'th action of the leader based on (weighted combination of) best response given by each follower type (where weight for each follower is its type probability)
end for
Choose the leader action which gives the best reward as the best deterministic strategy
```

The length of each parent or chromosome is equal to the *number of actions i.e., number of pure strategies of the leader* in GSG. Consider the sample normal form game defined in Table I. The leader in this game has two actions, hence each parent is of length two. A sample initial population of five parents for this game is shown in Table II:

i_1	0.5	0.5
i_2	0.65	0.35
i_3	0.15	0.85
i_4	0.9	0.1
i_5	0.7	0.3

TABLE II: Sample population

Note that i_1, i_2, i_3, i_4, i_5 in the table denote the chromosomes where i_1 = [0.5, 0.5], i_2 = [0.65, 0.35] and so on.

B. *Fitness function*

The fitness function needs to define how fit or good a candidate solution (parent) is for the problem under consideration. The objective of our problem is to identify the best possible mixed strategy for the leader. As stated earlier, the algorithm in [15] aims to compute Stackelberg equilibria where the payoff functions are continuous and differentiable. We make suitable modifications to their fitness function to adapt for discrete actions and payoff as presented below. In particular, we model a maximization function for the follower, given an arbitrary mixed strategy of the leader. We then use this 'functor' inside the objective function of the leader. This is also inline with idea behind game theoretic solution approaches for GSGs e.g., DOBSS algorithm [13], although there is no notion of a fitness function.

The fitness function for our purposes is computed as follows: For a given chromosome (i.e., mixed strategy of the leader) picked from the population, the fitness function starts by computing the reaction of the follower to the chromosome (i.e., the leader's strategy). It then evaluates the leader's strategy, taking into the account the follower's reaction to this strategy of the leader. An inner optimization problem (corresponding to the follower) is solved each time the GA evaluates the leader's strategy. Hence, we model the fitness evaluation as a bilevel optimization problem [10], [11]. Using the following notation, the fitness function can be mathematically represented as follows: Let the leader be row player and follower be column player. We denote X as index set of pure strategies for the leader, L denotes the set of follower types and Q^l denotes the index set of pure strategies for follower of type l. R^l_{ij} and C^l_{ij} are the rewards of the leader and the follower of type l respectively. p^l denotes the a priori probability that a follower of type l will appear and Z_i denotes the probability (i.e., value between 0 to 1) of using pure strategy i, from the index set X of the leader.

$$\begin{aligned} \max \quad & \sum_{i\in X}\sum_{l\in L}\sum_{j\in Q^l} p^l R^l_{ij} Z_i \\ \text{s.t.} \quad & \\ & j \in \underset{j\in Q^l}{\operatorname{argmax}}\{\sum_{i\in X} C^l_{ij} Z_i\} \\ & \sum_{i\in X} Z_i = 1 \\ & Z_i \in [0,1] \end{aligned} \tag{1}$$

The $argmax$ function computes the best response j of follower, for a given mixed strategy of the leader (as encoded in a chromosome of the population). The computed response j is then used in the objective function, to compute the maximum expected reward for the leader (i.e., fitness value). Table III showcases fitness values for the population defined in Table II, along with the Best Response (BR) of the follower for each instance. For example, row 1 of Table III corresponds to the fitness of chromosome i_1 and is computed as follows: For the game with X = [a,b], L = [1] (only 1-follower type), Q^l = [c,d] and payoffs for leader and follower as defined in $Table\ I$, for a given mixed strategy (i_1 with probabilities [0.5, 0.5] here), the fitness function computes the follower's reaction and then evaluates the reward (i.e., fitness value), that can be obtained for the leader using that strategy. The follower's best reaction (i.e., Best Response) for i_1 is d, since it ensures a higher reward of 1.0 for itself compared to a reward of 0.5 when it chooses c. Hence, the *max* reward (i.e., fitness value) for the leader would be 3.5 for i_1.

parent	*fitness*	*BR*
i_1	3.5	d
i_2	3.65	d
i_3	3.15	d
i_4	2	c
i_5	1.7	c

TABLE III: Example for fitness

C. *Selection*

The selection operation involves selection of parents for the next generation. We use the Tournament selection [18] method for our purposes, which is widely used due to a number of properties including its efficiency, low susceptibility to takeover by dominant parents, and simple implementation. In tournament selection, n parents (i.e., mixed strategies for leader) are selected randomly from the larger population, and the selected parents compete against each other. The parent with the highest fitness (i.e., mixed strategy that gives highest leader reward) wins and will be included in the next generation population. The number of parents competing in each tournament is referred to as tournament size. Tournament selection provides a chance for all parents to be selected and thus it preserves diversity, although keeping diversity may degrade the convergence speed.

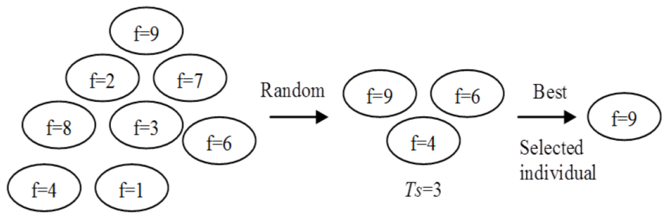


Fig. 1: Selection strategy with tournament mechanism

Figure 1 adapted from [18], shows an example for the tournament selection procedure. The figure showcases a population of size 8 with each parent assigned a fitness score

(f). The tournament size (T_s) in the figure is 3. Hence, three chromosomes are selected randomly from the population of size 8 in an iterative fashion where the chromosome with best fitness is selected in each iteration e.g., chromosome with fitness score of 9 is selected in the first iteration. The chromosome selected at each iteration will be included in the parent set and the process is repeated for n iterations, to identify n parents for the crossover and the mutation operations.

D. Crossover

Crossover operation is also called *recombination* and is used to combine the genetic information of two parents to generate new offspring(s).The crossover operator we use in this paper is a standard one called Simulated Binary Crossover(SBX) [19]. In particular, we use a bounded variant of it as presented in [19] with a lower bound of 0 and upper bound of 1. Reader can refer to DEAP Framework [21] and the appendix of [22], for implementation details of this procedure.

E. Mutation

As stated earlier in Introduction, we use a custom mutation operator which is a combination of an exhaustive search and its relaxed version. Although it is comprised of two search methods, only one of them is used based on a random number (between 0 and 1). The pseudo code for the mutation operator is shown in Algorithm 3.

Algorithm 3 Mutation

```
if random(0,1) $< 0.3$ then
    mutation1()                    ▷ exhaustive search
else
    mutation2()    ▷ relaxed variant of exhaustive search
end if
```

The reason for this approach is as follows: we observed via our experiments that the exhaustive mutation operator (mutation1) takes a long computation time. The relaxed variant on the other hand (mutation2), can result in a lower fitness. We therefore use a combination of these two variants, with the intent to obtain high enough fitness values while being reasonable in terms of computation time. We present descriptions of these operators below.

Algorithm 4 showcases the exhaustive search used in the mutation process (mutation1) to find a better individual. We first define a set of δ's, which are the possible values by which we vary the probability of an action. For a given vector/mixed strategy, we iteratively go over each action and δ and we add δ to the action's probability. All possible combinations of actions and δ's are tested to find the best possible vector. Whenever a better vector is found, we store the new vector as potential best and continue testing with other action-δ combinations, till all the combinations are tested. Please note that whenever a δ value is added, the sum of probabilities of actions of the new vector would not add up to 1. We therefore normalize the new vector at every combination and compare it with the best vector found till that stage. The set of values for delta, we used for our experimentation are [0.05,0.1,0.25,0.5]. We have experimented with different values of δ's to arrive at the values assigned.

Algorithm 4 mutation 1

```
$\delta = [\delta_1,\delta_2,\delta_3 \ldots ,\delta_n]$
benefit = 1
while benefit $> 0$ do
    benefit $= 0$
    for i =0 to Number of actions of leader do
        for k =0 to Number of deltas do
            create new normalised vector (i-e..,mixed strategy/chromosome) by adding $\delta_k$ to $i^{th}$ action
            local_benefit= value(new_vec)-value(old_vec)
                ▷ difference in fitness/reward between new created vector and old one
            if local_benefit $>$ benefit then
                benefit = local_benefit
                old_vec $=$ new_vec          ▷ chromosome is updated
            else
                vector remains unchanged
            end if
        end for
    end for
end while
```

In the relaxed version of mutation1() i.e., mutation2() (as referenced in algorithm3), we do the following instead of testing every possible combination of δ's and actions: We first pick an action randomly. We then pick a δ randomly and check if the new vector formed by adding the δ to that action probability, increases the fitness of the vector. If the new vector is better we terminate the mutation, else we randomly pick another value for δ (repetition not allowed) and continue testing. If we do not find a better vector after testing with all δ's for that action, we terminate the mutation. As shown in algorithm 3, mutation2() gets picked most times unless random number generated is less than 0.3 (with an aim to keep the computation overheads low).

F. Normalization

We define population as a set of chromosomes. Each chromosome represents a mixed strategy, i.e., probability distribution over the set of actions/pure strategies of the leader. The crossover and mutation operations can result in probability values that may not sum to 1 for a chromosome. Hence, chromosomes are normalized during crossover and mutation whenever needed, to satisfy the following constraints [23]: (a) Probability assigned to each action must be between $[0, 1]$ (b) Sum of the probabilities across all actions must sum to 1.

G. Replacement policy

At the end of each generation, if the new generation of offsprings are less fit than each of their respective parents, they

TABLE IV: GA Parameter details

Parameter	Value
Population size	50
Crossover rate	0.9
Mutation rate	0.1
Selection	Tournament Selection with selection size=3
Crossover mode	Simulated Binary Bounded Crossover
Lower bound $x^{(L)}$	0
Upper bound $x^{(U)}$	1
Distribution index η	0.1
Mutation mode	custom mutation
Generations	100

are not allowed in the next generation. From perspective of crossover, if $c1$ and $c2$ are the off-springs/children generated when crossover is performed with $p1$ and $p2$ as parents, $c1$ is considered as child of $p1$ and $c2$ is considered as child of $p2$. No such notation is needed for mutation since a parent generates only one child. We borrow this replacement policy from [24], to improve the speed of convergence.

H. Termination Conditions

The GA terminates if any of the following conditions are met:

1) GA reaches 100 generations.
2) The time limit is crossed (which we used as 1 hour (3600 seconds) in our experiments)
3) The standard deviation (of fitness) of the population [25] is less than 1×10^{-4}
4) The difference between the fitness value of the best chromosome in the current generation and the best fitness value from previous generation is less than 1×10^{-4} for 10 consecutive generations.

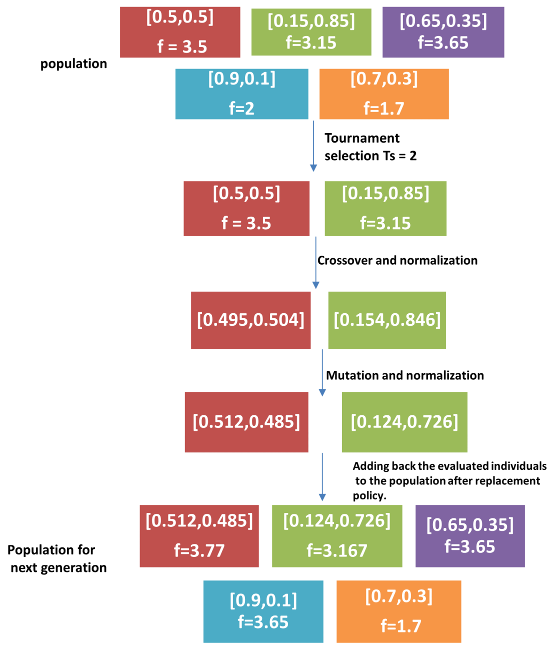


Fig. 2: Sample GA process for one generation

III. Example working of GA for one iteration

Figure 2 showcases one iteration of how our GA operates. Initially, we have a population of size 5 (i.e., five mixed strategies), each having a fitness value (f) shown in the figure. Two chromosomes are selected using tournament selection with tournament size T_s equal to 2. The two selected chromosomes are sent to crossover and then to mutation operations. Their fitness is computed and will replace the old chromosomes, if their fitness values are better than fitness of the old chromosomes using the process described in subsection II-G (e.g., if chromosomes $p1$ and $p2$ generate $c1$ and $c2$ via crossover, $c1$ is compared to $p1$ and $c2$ to $p2$ post performing mutation on $c1$ and $c2$ respectively). Note that at each generation we save the top 10% of chromosomes for next generation while the rest of them will undergo tournament selection (i.e., elitism strategy). The selected pool of chromosomes are then sent for crossover and mutation. For explanation purposes, if there are 50 chromosomes in initial population, we preserve the top 5 chromosomes (10%) while rest of them undergo tournament selection. The selected chromosomes along with the preserved 10%, are then sent to crossover and mutation operators.

IV. Experimental Domain

We use the domain presented in [26] as our domain, which is motivated by a patrolling and security application [27], [28] modeled as a game. As described in detail in [26], the game consists of two players namely security agent (leader) and robber (who is a follower) in a world consisting of m houses $1...m$. The security agent's set of pure strategies consists of possible routes of d houses to patrol (in some order). The security agent can choose a mixed strategy so that the robber will be unsure of exactly where the security agent may patrol, but the robber will know the mixed strategy the security agent has chosen e.g., the robber can observe over time how often the security agent patrols each house and choose a single house to rob. If the house chosen by the robber is not on the security agent's route, the robber successfully robs it. If not, since the robber takes time to rob a house, the earlier the house is on the agent's route, the easier it is (or higher the chance) for the security agent to catch the robber before the robbery is done. As presented in [26], the payoffs are modeled using the following variables:

- $v_{y,x}$:value of the goods in house y to the security agent.
- $v_{y,q}$:value of the goods in house y to the robber.
- c_x:reward obtained by the security agent for catching the robber.
- c_q:cost to the robber for getting caught.
- p_y:probability that the security agent can catch the robber at the yth house in the patrol ($p_y < p_{y'} \iff y' < y$)

The security agent's set of possible pure strategies (patrol routes) is denoted by X and includes d-tuples $i = < w_1, w_2, \ldots, w_d >$. Each of $w_1 \ldots w_d$ may take values 1 through m (different houses), however, no two elements of the d-tuple are allowed to be equal (the agent is not allowed to return to the same house). The robber's set of possible pure strategies (i.e., house to rob) is denoted by Q and each pure strategy denotes one of integers j = 1 . . . m. The payoffs for (security agent, robber) for pure strategies i, j are:

- -$v_{y,x}$, $v_{y,q}$, for $j = l \notin i$.
- $p_y c_x$ + (1 − p_y)(-$v_{y,x}$), -$p_y c_q$ + (1 − p_y) ($v_{y,q}$), for j = $y \in i$.

The above structure enables to model different types of robbers who have differing motivations e.g., one robber may have a lower cost of getting caught or may value the goods in the various houses differently. We use a probability distribution with varying probabilities for the different types of robbers. All the games are normalized so that the minimum and maximum payoffs are 0 and 1 both for the security agent and the different robber types.

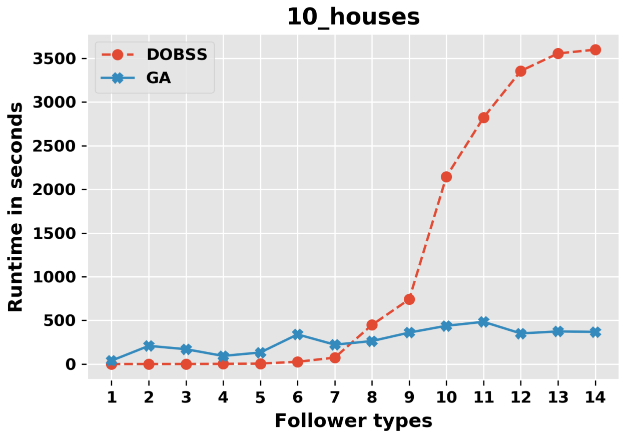
V. Experimental Results

We created games with 10 and 20 houses with games involving 1 to 14 follower types for 10-houses and 1 to 8 follower types for 20-houses. Each game models a patrol route consisting of two houses and five instances of each game setting were generated for averaging purposes. A route with patrol size of two houses translates to 90 actions for the leader in a game with 10 houses i.e., 10 P 2 (10 Permute 2) since order of houses patrolled matters. Each follower type in this case has 10 actions since the follower robs any one house. In a similar manner, the leader has 380 actions when the number of houses is 20 and each follower type has 20 actions. The payoff tables were generated using the method described in section IV. For a game with 20 houses and 8 follower types, we generate 8 payoff matrices each of size $380 * 20$.

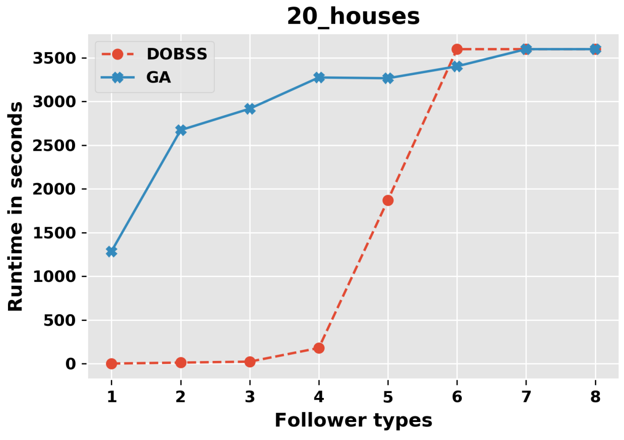
DOBSS is a popular algorithm for benchmarking purposes [6], [8], [13] and is used in this work to compute the optimal GSG solution. We used GUROBI 9.0 optimizer to implement the DOBSS algorithm and used DEAP 1.3 to implement our GA. Both the algorithms have been implemented with Python interface on a machine with i5 processor and 16 GB RAM. The cutoff time used for experiments is 1 hour (3600 seconds). In the case of DOBSS, the best deterministic strategy of the leader is used (computed using Algorithm 2), if the optimal mixed strategy could not be computed within the cutoff time. We refer to this version of DOBSS as DOBSS+DET. Our proposed GA is an anytime algorithm as we are using normalization (subsection II-F) to keep the chromosomes with the constraints. So, the best solution found by the time of cutoff is used. Regular termination conditions (apart from cutoff time), were described earlier in subsection II-H.

A. Runtime Results

Figure 3 shows the averaged runtime results for DOBSS and GA over five instances for each of the two settings involving 10 and 20 houses. Please note that DOBSS and DOBSS+DET have similar runtime, since computing the optimal deterministic strategy using Algorithm 2 needs less than a second. Hence, DOBSS+DET is not represented in the figure. The x-axis of each plot in the figure shows the number of follower types ranging from 1 to 14 for 10 houses and 1 to 8 for 20 houses. The y-axis for the plots shows the runtime measured in seconds. DOBSS has an exponential increase in runtime as the number of follower types increase whereas GA has a slowly increasing runtime with some uneveness. In both the settings, GA starts off needing more runtime initially with DOBSS crossing GA at some point. For example, in the plot for 10-houses with 14 follower types, DOBSS reached the limit of 1 hour (3600 seconds) and was not able to provide solution while the average runtime for GA with 14 follower types is 368.30 seconds. In the setting with 20-houses (a much bigger problem) both DOBSS and GA reach the cutoff limits at 6 and 7 follower types respectively. Given that our proposed GA is an anytime algorithm due to the usage of normalisation (so chromosomes fulfill constraints), it performs well in terms of scalability since it will have a solution identified at any point of time while DOBSS cannot provide a solution till it converges.



(a)



(b)

Fig. 3: Runtime plot for DOBSS and GA

B. Solution Quality Results

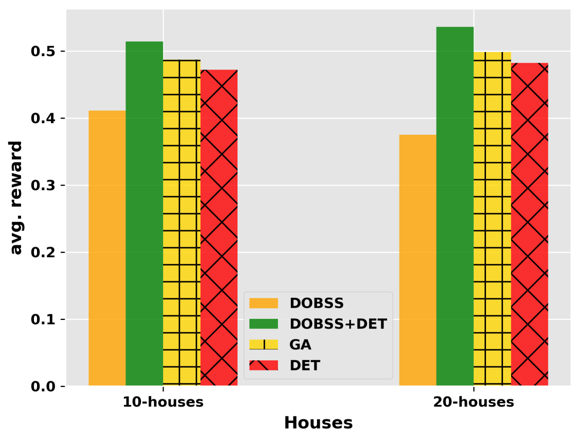
We now present the solution quality results for DOBSS, DOBSS+DET, GA and DET (best deterministic strategy) for the settings with 10 and 20 houses. As the proposed GA is an anytime algorithm, the best solution found by the time of cutoff is used. DOBSS being a mixed integer program, either identifies or does not identify a solution within the cutoff time in which case a value of zero is assigned for DOBSS and the reward obtained by the best deterministic strategy (DET) is used for DOBSS+DET.

Figure 4 showcases the average reward obtained using the different algorithms for the two settings involving 10 and 20 houses. Please note that the plots in the figure do not capture the number of follower types since the results are computed as follows: We first compute the average reward obtained by the leader over five instances for each setting of follower types i.e., from 1 to 14 follower types for 10-houses (corresponding to 14 leader rewards) and from 1 to 8 for 20-houses. We then compute the (overall) average reward for the leader as the average of rewards obtained across all the follower types i.e., if $l1$ is the leader reward for the setting with 1 follower type with 10 houses, $l2$ for the setting with 2 follower types and so on till ln for n follower types, the averaged reward for the setting with 10 houses is obtained as an average over $l1$, $l2$ till ln. Similar computation if performed for the setting with 20 houses.

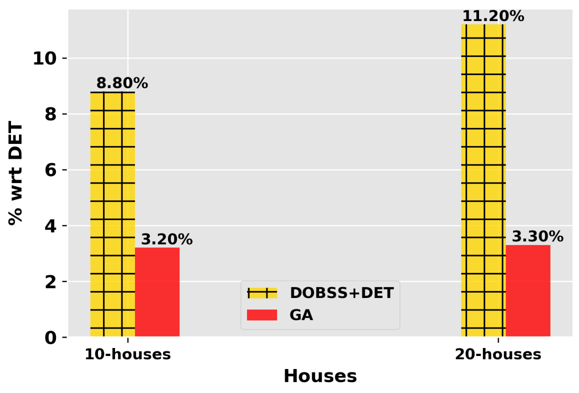
The first plot of Figure 4 (plot (a)), shows the average reward obtained for the settings with 10 and 20 houses. For the setting with 10 houses, the average reward obtained by DOBSS is 0.411, DOBSS+DET is 0.514, 0.487 for GA while the average reward is 0.472 for DET. Similarly, for the setting with 20 houses the average reward for DOBSS is 0.375, 0.536 for DOBSS+DET, 0.498 for GA while the value is 0.482 for DET. The second plot of the figure, shows the percentage difference of DOBSS+DET and GA w.r.t. DET and is computed as follows:

$$\%\ Diff = \frac{Algo - DET}{DET} * 100 \tag{2}$$

The second plot (i.e., plot (b)), shows that the average reward obtained by DOBSS+DET is 8.8% and 11.20% higher than DET for 10 and 20 houses respectively. Similar results for GA stand respectively at 3.2% and 3.3% higher than DET. This shows that our genetic algorithm provides reasonable improvements over DET which captures the best deterministic strategy for a GSG. In addition, as shown in plot (a), GA performs significantly better than DOBSS since DOBSS was unable to compute a solution within the cutoff time in a number of cases. We therefore use a combination model DOBSS+DET and results for this model show that the GA solution we provide here (possibly) has scope for further improvement. Please note that a number of parameter designs have been tried as part of this work to develop a high quality GA solution and the best design identified has been presented here.



(a)



(b)

Fig. 4: Solution quality plot for DOBSS and GA

VI. Conclusions

In this paper, we present a Genetic Algorithm (GA) to compute high quality mixed strategy solution for the leader to commit to in a General Stackelberg Game (GSG), when it faces multiple types of followers with discrete payoff functions. This is different from prior work in GA literature that focus on developing GAs for a Stackelberg game with utilities defined using continuous mathematical functions. DOBSS is a popular game theoretic algorithm for benchmarking purposes and is used to compute the optimal GSG solution in this paper. Our experiments showcase that the GA algorithm developed here performs well in terms of scalability and provides reasonably good solution quality in terms of the average reward obtained. Our future work will explore ways to improve further the solution quality obtained using GA. We also plan to explore possibilities to tailor the GA to capture better the domain characteristics/constraints of specific applications that Stackelberg games have been applied for.

VII. Acknowledgements

We thank Kohli Center on Intelligent Systems, International Institute of Information Technology, Hyderabad for the generous support.

References

[1] H. Von Stackelberg, *Market structure and equilibrium*. Springer Science & Business Media, 2010.

[2] W. Krichene, J. D. Reilly, S. Amin, and A. M. Bayen, "Stackelberg routing on parallel networks with horizontal queues," *IEEE Transactions on Automatic Control*, vol. 59, no. 3, pp. 714–727, March 2014.

[3] D. Kar, T. H. Nguyen, F. Fang, M. Brown, A. Sinha, M. Tambe, and A. X. Jiang, "Trends and applications in stackelberg security games."

[4] G. Brown, M. Carlyle, J. Salmerón, and K. Wood, "Defending critical infrastructure," *Interfaces*, vol. 36, no. 6, pp. 530–544, 2006.

[5] J. Pita, M. Jain, J. Marecki, F. Ordóñez, C. Portway, M. Tambe, C. Western, P. Paruchuri, and S. Kraus, "Deployed armor protection: the application of a game theoretic model for security at the los angeles international airport." in *AAMAS (Industry Track)*, 2008, pp. 125–132.

[6] M. Tambe, *Security and game theory: algorithms, deployed systems, lessons learned*. Cambridge university press, 2011.

[7] D. Fudenberg and J. Tirole, *Game Theory*. MIT Press, 1991.

[8] C. Casorrán, B. Fortz, M. Labbé, and F. Ordóñez, "A study of general and security stackelberg game formulations," *European journal of operational research*, vol. 278, no. 3, pp. 855–868, 2019.

[9] V. Bucarey, C. Casorrán, M. Labbé, F. Ordoñez, and O. Figueroa, "Coordinating resources in stackelberg security games," *European Journal of Operational Research*, 2019.

[10] A. Sinha, P. Malo, and K. Deb, "A review on bilevel optimization: From classical to evolutionary approaches and applications," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 2, pp. 276–295, 2017.

[11] B. Colson, P. Marcotte, and G. Savard, "An overview of bilevel optimization," *Annals of operations research*, vol. 153, no. 1, pp. 235–256, 2007.

[12] V. Conitzer and T. Sandholm, "Computing the optimal strategy to commit to," in *Proceedings of the 7th ACM conference on Electronic commerce*. ACM, 2006, pp. 82–90.

[13] P. Paruchuri, J. P. Pearce, J. Marecki, M. Tambe, F. Ordonez, and S. Kraus, "Playing games for security: An efficient exact algorithm for solving bayesian stackelberg games," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 895–902.

[14] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992.

[15] J. P. Pedroso *et al.*, "Numerical solution of nash and stackelberg equilibria: an evolutionary approach," in *Proceedings of SEAL*, vol. 96, 1996, pp. 151–160.

[16] E. D'Amato, E. Daniele, L. Mallozzi, and G. Petrone, "Equilibrium strategies via ga to stackelberg games under multiple follower's best reply," *International Journal of Intelligent Systems*, vol. 27, no. 2, pp. 74–85, 2012.

[17] N. M. Alemdar and S. Sirakaya, "On-line computation of stackelberg equilibria with synchronous parallel genetic algorithms," *Journal of Economic Dynamics and Control*, vol. 27, no. 8, pp. 1503–1515, 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0165188902000696

[18] N. M. Razali, J. Geraghty *et al.*, "Genetic algorithm performance with different selection strategies in solving tsp," in *Proceedings of the world congress on engineering*, vol. 2, no. 1. International Association of Engineers Hong Kong, 2011, pp. 1–6.

[19] K. Kumar and K. Deb, "Real-coded genetic algorithms with simulated binary crossover: Studies on multimodal and multiobjective problems," *Complex syst*, vol. 9, pp. 431–454, 1995.

[20] K. Deb and K. Pal, "Efficiently solving: A large-scale integer linear program using a customized genetic algorithm," in *Genetic and Evolutionary Computation Conference*. Springer, 2004, pp. 1054–1065.

[21] F.-M. De Rainville, F.-A. Fortin, M.-A. Gardner, M. Parizeau, and C. Gagné, "Deap: A python framework for evolutionary algorithms," in *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, ser. GECCO '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 85–92.

[22] K. Deb, "An efficient constraint handling method for genetic algorithms," *Computer methods in applied mechanics and engineering*, vol. 186, no. 2-4, pp. 311–338, 2000.

[23] C. A. C. Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art," *Computer methods in applied mechanics and engineering*, vol. 191, no. 11-12, pp. 1245–1287, 2002.

[24] Y.-C. Chuang, C.-T. Chen, and C. Hwang, "A real-coded genetic algorithm with a direction-based crossover operator," *Inf. Sci.*, vol. 305, no. C, p. 320–348, Jun. 2015.

[25] K. Q. Zhu, "Population diversity in genetic algorithm for vehicle routing problem with time windows," in *European Conference on Machine Learning. Pisa, Italy*, 2004, pp. 537–547.

[26] P. Paruchuri, J. P. Pearce, M. Tambe, F. Ordonez, and S. Kraus, "An efficient heuristic approach for security against multiple adversaries," in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, 2007, pp. 1–8.

[27] R. W. Beard and T. W. McLain, "Multiple uav cooperative search under collision avoidance and limited range communication constraints," in *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, vol. 1. IEEE, 2003, pp. 25–30.

[28] S. Ruan, C. Meirina, F. Yu, K. R. Pattipati, and R. L. Popp, "Patrolling in a stochastic environment," Tech. Rep., 2005.