Algorithm 1 Algorithm to learn the weights of P-SUQR and its variations in repeated Stackelberg games

INPUT: Data from R rounds: D^1, D^2, ... , D^R.
OUTPUT: Learned weights (δ_p, γ_p, ω_1, ω_2, ω_3, ω_4).

1: **for** r=1 to R **do**
2: Randomly divide the collected data D^r into 1 training (Tr^r) and 1 test (Te^r) set.
3: Take the training samples (Tr^r) and randomly divide it into K training (${}^kTrv^r$) and validation (${}^kVal^r$) splits ($1 \leq k \leq K$).
4: **end for**
5: Consider a range of values for both δ and γ (Eqn. 6 in the paper).
6: Discretize each range and consider all possible $\{\delta\ ,\ \gamma\}$ pairs in that range. Let there be M such pairs.
7: **for** i=1 to M **do**
8: **for** k=1 to K **do**
9: Given training splits ${}^kTrv^1$, ${}^kTrv^2$, ... , ${}^kTrv^R$, learn the weights ${}^k\omega$=(${}^k\omega_1$, ${}^k\omega_2$, ${}^k\omega_3$, ${}^k\omega_4$) of Eqn. 10 in the paper by using MLE to maximize the sum of log-likelihoods over all such training splits [1].
10: Predict using the learned weights ${}^k\omega$ on the corresponding validation splits ${}^kVal^1$, ${}^kVal^2$, ... , ${}^kVal^R$.
11: Calculate the prediction errors ${}^kErr^1$, ${}^kErr^2$, ... , ${}^kErr^R$ on the validation sets ${}^kVal^1$, ${}^kVal^2$, ... , ${}^kVal^R$ respectively.
12: Calculate the sum of all prediction errors ${}^kErr^1$, ${}^kErr^2$, ... , ${}^kErr^R$ and let it be ${}^kErr$.
13: **end for**
14: Calculate the average of all K prediction errors ${}^kErr$ ($1 \leq k \leq K$) and let that be denoted by $AvgErr_i$.
15: **end for**
16: Let p be the index of the $\{\delta\ ,\ \gamma\}$ pair with the minimum $AvgErr_i$ (i=1 to M). Choose $\{\delta_p\ ,\ \gamma_p\}$ as the parameter values of the probability weighting function that best describes the probability weights of the adversary population.
17: Given training sets Tr^1, Tr^2, ... , Tr^R and $\{\delta_p\ ,\ \gamma_p\}$, learn the weights ω=(ω_1, ω_2, ω_3, ω_4) of Eqn. 10 in the paper by using MLE to maximize the sum of log-likelihoods over all such training splits [1]. The final learned weight set is then (δ_p, γ_p, ω_1, ω_2, ω_3, ω_4).

Table 1: Performance (Squared Errors) of various feature sets

	Eqn. 7	Eqn. 8	Eqn. 9	Eqn. 10
P-SUQR ADS_1 Algorithm 1	0.1965*	0.2031*	0.1985	**0.1025***
P-SUQR ADS_1 Non-linear Solver	0.2545	0.2589	0.2362	**0.1865**
P-SUQR ADS_2 Algorithm 1	0.2065*	0.2156*	0.2625	**0.1136***
P-SUQR ADS_2 Non-linear Solver	0.2546	0.2935	0.3062	**0.1945**

To test the performance of Algorithm 1 against a non-linear solver (Microsoft Excel's Generalized Reduced Gradient (GRG) nonlinear solver function), we learn the weights of the four behavioral models (Eqn. 7 to 10 in the paper) using both Algorithm 1 and our non-linear solver. We divided the first round data for the experiment with P-SUQR on ADS_1 into 10 random train-test splits. We then computed the sum of squared errors (SE) of predicting the attack probability for each of the test splits and for each of the feature combinations using the weights learned by Algorithm 1 and the non-linear solver. We report the average SE for both the weight learning approaches on all the feature combinations in Table 1. Results accompanied by * imply significant differences in performance of Algorithm 1 as compared to the non-linear solver. Thus, Algorithm 1 is more efficient in learning model weights as compared to a non-linear solver.

References

[1] T. H. Nguyen, R. Yang, A. Azaria, S. Kraus, and M. Tambe. Analyzing the effectiveness of adversary modeling in security games. *In AAAI*, 2013.