2009 International Conference on Computational Science and Engineering

Tuning Evidence-Based Trust Models

Eugen Staab and Thomas Engel
University of Luxembourg
6, r. Richard Coudenhove-Kalergi
L-1359 Luxembourg
Email: {eugen.staab,thomas.engel}@uni.lu

***Abstract*—Many evidence-based trust models require the adjustment of parameters such as aging- or exploration-factors. What the literature often does not address is the systematic choice of these parameters. In our work, we propose a generic procedure for finding trust model parameters that maximize the expected utility to the trust model user. The procedure is based on game-theoretic considerations and uses a genetic algorithm to cope with the vast number of possible attack strategies. To demonstrate the feasibility of the approach, we apply our procedure to a concrete trust model and optimize the parameters of this model.**

I. Introduction

Evidence-based trust models use past experience to decide about the trustworthiness of potential interaction partners. Positive experiences generally increase the estimate of the trustworthiness, and negative experiences reduce it. How this is done in detail is specified by the numerous trust models that have been proposed in the literature (e.g. see [1]–[3]). Many of these trust models can be configured using one or more parameters. For instance, some models use an *aging factor* that determines the weight given to older experiences. Other models use a specific threshold to decide when an opponent is thought to be trustworthy enough to be relied upon.

Most often, the effectiveness of the proposed trust models is evaluated for specific parameter settings (configurations) of the models. While such an evaluation can show how well the model works for these specific configurations, the evaluations do not tell us whether the model could perform even better if the configuration were improved. Unfortunately, it is often left open how systematically to find suitable configurations for the proposed trust models. Therefore, methods for optimizing the parameters of a trust model are required.

In this work, we propose a generic method for finding optimal configurations for evidence-based trust models. The idea is to consider the deployment of a trust model as a *game* against a malicious opponent – the *attacker*. By applying a game-theoretic solution concept, we can assess which configuration of the trust model would yield the highest utility. Our solution concept can deal with a payoff matrix that is only partially known. However, this approach requires knowledge of the strategy an attacker is most likely to choose for a given trust model configuration. We show how such strategies can be determined by searching in the space of possible attack strategies. Since the number of attack strategies is infinite, we propose a genetic algorithm that can cope with this huge search space.

In the second part of the paper we apply our tuning procedure to a concrete trust model that is designed for result verification in *desktop grids* (e.g., SETI@home [4]). As a result, we find a trust model configuration that will provide the highest expected utility against rational attackers.

Organization: The remainder of this paper is organized as follows. In Sect. II, we model the deployment of a trust model as a game. We present the procedure for optimizing the trust model configuration in Sect. III. In Sect. IV, the proposed approach is applied to tune a concrete trust model. We discuss the approach in Sect. V. Finally, we review related work in Sect. VI and draw conclusions in Sect. VII.

II. Trust Models in Games

A. Formalization

We interpret the deployment of a trust model as a game between the trust model user, called the "trustor", against a set of attackers and honest players. A game consists of a sequence of a fixed number of rounds. In each round, the trustor selects an opponent and requests some service σ. If the selected opponent is honest, he acts in a predefined way (in Sect. IV we will give more details on this). If the opponent is one of the attackers, he can respond with some action α chosen from a set of actions $\mathcal{A}$.

The finite strategy space S_T contains all possible configurations of the considered trust model[1]. So, a strategy $s_T \in S_T$ determines for a given game history how the trustor decides which service σ to request from which opponent in the next round. The possibly infinite strategy space S_A of an attacker contains any attack strategy that can be deployed against the respective trust model. These strategies describe which action α an attacker will choose for a given game history and a requested service σ.

For each pair (σ, α), two functions $\mathtt{U}_T$ and $\mathtt{U}_A$ define, for the trustor and the attacker respectively, the real-valued utilities for a single round where the trustor requests σ, and the opponent responds with α. Another utility function $\mathcal{U}_T : S_T \times S_A \mapsto \mathbb{R}$ defines the expected utility of the trustor for an entire game. Analogously, a function $\mathcal{U}_A : S_T \times S_A \mapsto \mathbb{R}$ defines the expected utility of an attacker for an entire game. Table I illustrates the payoff matrix containing the expected utilities for an entire game. Rows represent strategies of the trustor

[1]To ensure that S_T is finite, parameters of the trust model have to be bound if necessary, and continuous parameters have to be discretized.

978-0-7695-3823-5/09 $26.00 © 2009 IEEE
DOI 10.1109/CSE.2009.209

TABLE I
PAYOFF MATRIX FOR AN ENTIRE GAME

	s_{A1}	s_{A2}	$\dots$
s_{T1}	$\mathcal{U}_T(s_{T1}, s_{A1})$, $\mathcal{U}_A(s_{T1}, s_{A1})$	$\mathcal{U}_T(s_{T1}, s_{A2})$, $\mathcal{U}_A(s_{T1}, s_{A2})$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\ddots$
s_{Tn}	$\mathcal{U}_T(s_{Tn}, s_{A1})$, $\mathcal{U}_A(s_{Tn}, s_{A1})$	$\mathcal{U}_T(s_{Tn}, s_{A2})$, $\mathcal{U}_A(s_{Tn}, s_{A2})$	$\dots$

TABLE II
PAYOFF MATRIX FOR AN ENTIRE GAME WHEN CONSIDERING ONLY BEST RESPONSES br*

	$\mathrm{br}^*(s_{Ti})$
s_{T1}	$\mathcal{U}_T(s_{T1}, \mathrm{br}^*(s_{T1}))$, $\mathcal{U}_A(s_{T1}, \mathrm{br}^*(s_{T1}))$
$\vdots$	$\vdots$
s_{Tn}	$\mathcal{U}_T(s_{Tn}, \mathrm{br}^*(s_{Tn}))$, $\mathcal{U}_A(s_{Tn}, \mathrm{br}^*(s_{Tn}))$

(trust model configurations), and columns represent attack strategies. We write (S_T, S_A)-*game* for a game where a trustor and an attacker can choose from strategy spaces S_T and S_A respectively. Analogously, we write (s_T, s_A)-*game* for a game, where a trustor and an attacker play strategies s_T and s_A respectively.

B. Stackelberg Competition

We analyze an (S_T, S_A)-game in the form of a *Stackelberg* competition (see [5, pp.67–69]). In a Stackelberg competition, the first player, the so-called "leader", chooses his strategy first. The second player, the "follower", can then choose his strategy with the knowledge of the leader's strategy. In our case, the trust model user represents the leader, who has to choose a trust model configuration $s_T \in S_T$ as his strategy. The attacker takes the role of the follower and can choose a strategy $s_A \in S_A$ that is a (best) response to the concrete configuration of the trust model. It is reasonable to consider the deployment of a trust model as a Stackelberg competition, since the attacker can in general observe and analyze a trust model for an extended period before he eventually attacks the model under a different identity.

Stackelberg competitions originally address the situation where two firms compete by varying output levels, and so the payoffs of leader and follower are usually linked linearly. In contrast, for the trust model deployment we want to consider all kinds of payoff matrices. We will see in the next section that a similar reasoning as for the original Stackelberg competition is still possible.

III. CONFIGURATION PROCEDURE

In this section we describe the procedure for tuning the parameters of an evidence-based trust model. In part III-A, we detail a game-theoretic solution concept that yields a Nash equilibrium [6]. We then describe the tuning procedure in parts III-B and III-C.

A. Selecting a Nash Equilibrium

In what follows, we assume *perfect rationality* of the players: they know how to maximize their utilities and act accordingly. Since there might be several best responses by an attacker to a trust model configuration, we assume the worst case for the trustor and let the attacker, in a second step, minimize the trustor's utility. This is reflected by the following notation.

Notation 1: For an (S_T, S_A)-game and $s_T \in S_T$, $\mathrm{br}^*(s_T) \in S_A$ denotes a best response to strategy s_T that causes for T the worst utility among all best responses to s_T (if there are several). Formally, for all s_T and any $\mathrm{br}^*(s_T)$ it holds:

$$\forall s_A \in S_A . \big[u_A(s_T, s_A) \leq u_A(s_T, \mathrm{br}^*(s_T))\big] \quad (1)$$
$$\wedge \big[u_A(s_T, s_A) = u_A(s_T, \mathrm{br}^*(s_T)) \quad (2)$$
$$\Rightarrow u_T(s_T, s_A) \geq u_T(s_T, \mathrm{br}^*(s_T))\big] . \quad (3)$$

Using this notation, we define a new solution concept similar[2] to the *Stackelberg outcome* [5] as follows:

Definition 1 (Stackelberg outcome):* Assuming a rational attacker in a (S_T, S_A)-game, strategy pair $(s_T, \mathrm{br}^*(s_T))$ is a *Stackelberg* outcome* if there is no other strategy $s'_T \in S_T$ for which the following holds:

$$u_T(s'_T, \mathrm{br}^*(s'_T)) > u_T(s_T, \mathrm{br}^*(s_T)) . \quad (4)$$

In other words, a leader will choose a configuration s_T that maximizes his utility knowing that a rational follower's strategy is to play $\mathrm{br}^*(s_T)$.

Theorem 1: A Stackelberg* outcome constitutes a Nash equilibrium.

To prove Theorem 1, we have to show that for a Stackelberg* outcome, players cannot get greater utilities by unilaterally changing their strategy.

Proof: Let $(s_T, \mathrm{br}^*(s_T)) \in S_T \times S_A$ be a Stackelberg* outcome. The trustor will not change his strategy to some different $s'_T \in S_T$ because he knows that the attacker would then play $\mathrm{br}^*(s'_T)$ which would not cause a higher utility for the trustor – because following Def. 1 we have:

$$u_T(s_T, \mathrm{br}^*(s_T)) \geq u_T(s'_T, \mathrm{br}^*(s'_T)) .$$

But the attacker also has no incentive to play a different strategy $s_A \in S_A$ with $s_A \neq \mathrm{br}^*(s_T)$, because his utility could not increase (see eq. 1). ■

B. Configuration

To find an optimal trust model configuration, we search for a Stackelberg* outcome. For this, we need to consider a restricted payoff matrix as shown in Table II. As before, rows represent the strategies of the trustor, i.e. trust model configurations. The only column now represents the best responses br^* to the corresponding trust model configuration. From this matrix the trustor selects the row strategy for which

[2]The actual difference is that we consider all kinds of payoff matrices and that the follower minimizes the leader's utility in a second step.

his expected utility is maximal. This yields a Nash equilibrium as shown in the previous section.

The restricted matrix assumes that we know the best responses of an attacker for each trust model configuration. In the following section, we show how to efficiently find best responses of an attacker in huge attack strategy spaces.

C. Finding Best Responses

We want to find the best response of an attacker to a given configuration of a trust model. However, the attacker can choose from a huge set of strategies, and so we do not want to play a game for every possible attack strategy to see how effective the strategy would be. The idea is therefore to search for the best response in the space of attack strategies. To this end, we employ a *genetic algorithm* that starts with a finite set of strategies, and lets the strategies evolve, until eventually a best response is found. In the following we will first specify the components of the genetic algorithm and then show how to define the attack strategy search space.

1) The Genetic Algorithm: A genetic algorithm is a stochastic optimization technique that is inspired by the process of natural selection (see [7], [8]). It starts with an initial population of individuals whose parameters are set randomly. By means of *selection*, *reproduction* and *mutation* the population evolves over several generations, and optimizes itself with respect to a fixed *fitness function*. The algorithm usually stops if either the individuals cease to improve, or a maximum number of generations is reached. The search is performed with several individuals to reduce the probability of getting trapped in local maxima [8].

For our purposes, we specify the different components of the genetic algorithm as follows:

a) Individual: The individuals represent the attackers. An individual is thus defined by a specific attack strategy. At the beginning of the algorithm each individual is initialized with a random attack strategy that is chosen from the attack strategy space.

b) Fitness Function: The fitness of an individual using a certain attack strategy s_A (against a trust model with configuration s_T) is given by the utility function $\mathcal{U}_A(s_T, s_A)$. The value of the utility function is determined by averaging over the utilities obtained in a statistically significant number of simulated (s_T, s_A)-games.

c) Selection: An individual is selected for reproduction with a probability that is proportional to the individuals' fitness. Parameter $\mathcal{S}$ specifies the *selection pressure*, which defines which impact (small) differences in the fitness have on the resulting probabilities. To be more specific, if I is the population of individuals and $f(i)$ is the fitness of individual $i \in I$, then we select i for reproduction with probability

$$\frac{e^{\mathcal{S}f(i)}}{\sum_{j \in I} e^{\mathcal{S}f(j)}} \,. \tag{5}$$

d) Reproduction: Given that two individuals have been selected for reproduction, a new individual is created that takes a part of the parameters from the "mother" and the other part from the "father". How this is done in a reasonable way depends on the form of the attack strategies. In our algorithm, mother and father are not necessarily part of the subsequent generation. We make sure that the size of the population does not change.

e) Mutation: For a certain fraction of the individuals that result from reproduction, one or more parameters are *mutated*, i.e. changed in a defined way. To a continuous parameter one can for instance add a random number following a gaussian distribution. Again, this depends on the form of the attack strategies.

f) Termination: The algorithm terminates if either of the following happens:

- the fitness of the best individuals of a certain number of consecutive generations does not change significantly,
- a certain number of generations has been reached. In our case, where the fitness function is computed by means of simulation, this points to the case that in an earlier generation an improbably high average utility was reached. As a countermeasure the number of games averaged over should be increased.

2) Attack Strategy Search Space: Let $\mathcal{A}$ be the set of all possible actions α an attacker can take in a single interaction. Then the attack strategy space for a game with r rounds is given by $\mathcal{A}^r$. This space contains however many inefficient attack strategies, and therefore we build the space by adding strategies to the empty set ($\emptyset$), instead of removing strategies from $\mathcal{A}^r$. To be more specific, we define a set of (sophisticated) basic strategies, each of which is parameterized with a finite set of parameters. The more comprehensive the set of sophisticated strategies is, the more likely it contains the actual best response. The drawback of this approach is that we might forget important attack strategies. As a consequence, the parameter settings selected by our tuning procedure would not be optimal. However, we clearly need to restrict the set $\mathcal{A}^r$, if r is large. Note that by adding only a selection of attacks to the search space, a trust model can be optimized specifically against these attacks.

In Sect. IV-B2, we will give an example for how the parameterized attack strategies can be defined. The resulting space is still very huge, which is why we employ the genetic algorithm for searching in it.

IV. Application

In this section, we use the above proposed procedure to optimize a trust model's configuration. The trust model that we consider is based on previous work [9]. We first describe this trust model and show which parameters can be optimized. Then, the utility functions for both trustor and attacker are specified and the implemented attack strategies are detailed. Finally, we present the results of the optimization procedure.

A. Trust Model with Uncertainty

The trust model allows for acquiring information from possibly untrustworthy sources. The model addresses the case where the correctness of the acquired information cannot

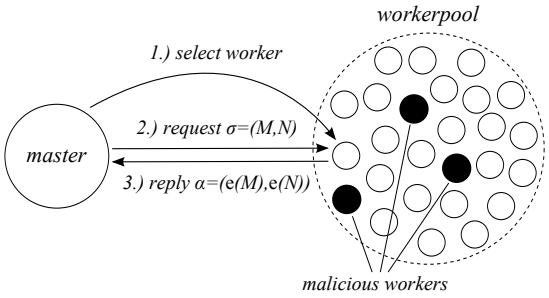

Fig. 1. Trust Model Scenario.

be verified easily. This problem exists in particular in the distributed computing domain, where computations are outsourced but the results could not be verified, because no efficient verification scheme is known for general computations [10]. More precisely, the trust model is intended for the use in desktop grid systems [11] where a *master* (the trustor) delegates work units to autonomic *workers* (who can be attackers).

The basic idea of the trust model is simple (see also Fig. 1). At the beginning, the master selects a worker to whom a request will be sent. Then, the master creates a request σ by merging a set M that contains m *work units* (the results to these will not be verified), with a set N that contains n *challenges*, i.e. work units for which the correct result is already known or can be determined. The master submits the request to the selected worker and awaits a response α. By verifying the results for the challenges, the master can estimate the error rate of the results for the remaining work units. For this, the model assumes that the correctness of the results is statistically independent, and that challenges are chosen in a way such that a worker is not able to distinguish them from real work units. Then the error rate of M, written $\mathrm{e}(M)$, can be estimated as follows: If $n^{\oplus}$ correct and $n^{\ominus}$ incorrect results to the challenges are found, the estimated error rate $\hat{\mathrm{e}}(M)$ for the work units in M is given by (see also [9]):

$$\hat{\mathrm{e}}(M) := \frac{n^{\ominus} + 1}{n^{\oplus} + n^{\ominus} + 2} \,. \tag{6}$$

In the considered scenarios, an error rate $t_{\mathrm{e}} > 0$ in M is acceptable. This is for example the case for applications in the field of image or video rendering, or statistics applications [12]. If the estimated error rate is higher than the tolerable error rate t_{e}, the entire response α is discarded.

Since the verification of a result of a challenge can be costly (in the grid scenario it is in general as costly as finding a result to a real work unit), we want to have $|N| \ll |M|$. However, the fewer challenges we use, the higher is the probability that we over- or underestimate the actual error rate of the results in M. At this point, we use *trust* as a concept to keep the number of challenges as small as possible. If the master has some evidence that a worker is trustworthy, he will reduce the number of challenges. This is detailed in the following.

1) Trust and Uncertainty Values: To estimate the trustworthiness of a worker, we first process the evidence we have collected so far with that worker.

Suppose we find $n^{\oplus}$ correctly computed challenges by worker w in interaction i, and suppose $a_{w,i-1}$ represents the positive evidence with worker w up to interaction i. Then we update the positive evidence with worker w for interaction i as follows:

$$a_{w,i} = A \cdot n^{\oplus} + (1 - A) \cdot a_{w,i-1} \,, \tag{7}$$

where A is the so-called *aging factor*. The aging factor is one of two parameters in our trust model. It determines how strongly older evidence shall be weighted in relation to more recent evidence.

Analogously we compute negative evidence with worker w based on the number $n^{\ominus}$ of incorrectly computed challenges in interaction i, and the negative evidence up to interaction $i - 1$:

$$b_{w,i} = A \cdot n^{\ominus} + (1 - A) \cdot b_{w,i-1} \,. \tag{8}$$

Before we have made any experience with worker w we set the initial evidence to $a_{w,0} = b_{w,0} = 0$.

Based on the positive and negative evidence, we define the trust value for worker w as the mean of a beta distribution (for details see [9]):

$$\mathrm{t}(w,i) := \frac{a_{w,i} + 1}{a_{w,i} + b_{w,i} + 2} \,. \tag{9}$$

Thus, the initial trust value for any worker is $\mathrm{t}(w,0) = 0.5$.

Our trust model computes also another value for each worker, called *uncertainty*. This value reflects how uncertain we are about a trust value. The uncertainty value for a worker w in interaction i is given by the normalized variance of a beta distribution (for details see [9]):

$$\mathrm{u}(w,i) := 12 \cdot \frac{a_{w,i} \cdot b_{w,i}}{(a_{w,i} + b_{w,i})^2 (a_{w,i} + b_{w,i} + 1)} \,. \tag{10}$$

2) Number of Challenges: In [9], we showed how a number of challenges can be found that is optimal in respect to the error estimation. We will use this optimal number as *minimum* number of challenges used by the trust model (unlike proposed in [9]). This way, no attacker can ever really harm the system.

To determine this optimal number of challenges one first has to define the costs that one challenge causes (c_c), and the costs that one erroneous work unit would cost (c_d). Then the expected costs can be computed as follows:

$$c(m,n) = n * c_c + m * c_d * d(m,n) \,, \tag{11}$$

where $d(m,n)$ is the expected estimation error when using n challenges to estimate the error rate of the m responses to the real work units (for details see [9]). To find the optimal number of challenges, $c(m,n)$ has to be minimized.

In our experiments, we set the costs to $c_c/c_d = 1/10$, and use in each round $m = 100$ work units. The expected costs $c(m,n)$ for this setting are shown in Table III for $m = 100$. So, the minimal number of challenges we should use is $n_{min} = 5$.

TABLE III
EXPECTED MISESTIMATION AND COSTS FOR $m = 100$.

n	$d(100, n)$	$c(100, n)$
1	0.200	21.048
2	0.171	19.127
3	0.152	18.212
4	0.138	17.841
5	0.128	17.797
6	0.120	17.968
7	0.113	18.294
8	0.107	18.726
9	0.102	19.251
10	0.098	19.837
⋮	⋮	⋮

However, this small n_{min} does not allow to fast learn about the trustworthiness of a worker. Therefore, we start with a rather high number of challenges ($n = m = 100$), and reduce this number as soon as we know the worker better. In order to determine how to reduce this number, we use the trust- and uncertainty values. For this, the following should hold:

- the more trustworthy a worker is thought to be, the less challenges are used, and
- the more uncertain one is about the trustworthiness estimate, the more challenges are used.

These criteria are met by the following formula:

$$n = n_{min} + (m - n_{min})(1 - t + ut) \; . \tag{12}$$

For $m > n_{min}$, the formula also guarantees that at least n_{min} and at most m many challenges are used.

3) Choosing a Worker: For selecting the next worker, we use the *boltzmann exploration* approach [13, pp.328–329]. In this approach, the probability for choosing worker w from a set of workers W in round i is given by (we use the trust value from eq. 9 as the estimated "value of the decision"):

$$P(w, i) = \frac{e^{B \cdot \mathrm{t}(w,i)}}{\sum_{v \in W} e^{B \cdot \mathrm{t}(v,i)}} \; , \tag{13}$$

where parameter B is the second parameter of our trust model. Basically, the smaller B is, the more impact the trust value has on the selection process. Note that we use the same approach in the selection process of the genetic algorithm (eq. 5).

However, the boltzmann exploration approach does not account for the amount of information we have collected so far, and how recent this information is. Therefore, we extend eq. 13 with the measure for uncertainty from eq. 10:

$$P'(w, i) = \frac{1 - \mathrm{u}(w, i)}{\sum_{v \in W} (1 - \mathrm{u}(v, i))} \cdot P(w, i) \; . \tag{14}$$

This weighting favors workers whose trustworthiness is more certain over those for which it is less certain.

B. Utilities and Attack Strategies

1) Utility Functions: Let $\sigma = (M, N)$ be a request that the master delegates to a certain worker. Furthermore, let $\alpha = (\mathrm{e}(M), \mathrm{e}(N))$ be the response of the worker, where $\mathrm{e}(M)$ and $\mathrm{e}(N)$ denote the error rates for the results to the real work units M and the challenges N respectively. Then we use the functions described in the following to compute in each round the utilities for the master and the chosen worker.

a) Master: Independently of the response of the worker, the master pays the costs of the challenges ($-n$). If the estimated error rate is acceptable (i.e., $\leq t_{\mathrm{e}}$), the master benefits from $m - m \cdot \mathrm{e}(M)$ correct results. This gives:

$$\mathtt{U}_T(\sigma, \alpha) = \begin{cases} -n, & \text{if } \hat{\mathrm{e}}(M) > t_{\mathrm{e}} \\ m - m \cdot \mathrm{e}(M) - n, & \text{if } \hat{\mathrm{e}}(M) \leq t_{\mathrm{e}} \end{cases} \tag{15}$$

The different terms could be weighted with different cost parameters, e.g., an incorrect result might cost more than the use of a challenge. However, to keep things simple, we set all costs to 1.

b) Attacker: If the response of the attacker is discarded, the attacker does not earn anything. If it is not discarded, he is attributed $m + n$ computations and saved resources for additional $m \cdot \mathrm{e}(M) + n \cdot \mathrm{e}(N)$ many work units. These "saved resources" can be spent on other work units and so are simply added to the utility. This gives:

$$\mathtt{U}_A(r) = \begin{cases} 0, & \text{if } \hat{\mathrm{e}}(M) > t_{\mathrm{e}} \\ m + n + m \cdot \mathrm{e}(M) + n \cdot \mathrm{e}(N), & \text{if } \hat{\mathrm{e}}(M) \leq t_{\mathrm{e}} \end{cases} \tag{16}$$

Note that in this context our initial trust value of 0.5 makes *whitewashing attacks* ineffective. In whitewashing attacks (usually known in the context of reputation systems [14]), workers with low trust values leave a system and reenter with a new identity, in order to "reset" the trust value (in our case to 0.5). However, in our case this only makes sense if the trust value of a worker dropped below 0.5; and in order to get $t < 0.5$ a worker must have returned a response with an error rate $\hat{\mathrm{e}}(M) > 0.5$ – but already for a detected error rate $\hat{\mathrm{e}}(M) > 0.1$, a worker gets no payoff (see eq. 16) and so has no incentive to act in this way.

2) Attack Strategies: Attack strategies against the trust model from Section IV-A consist in determining for each interaction the ratio of wrong results in the response α. We modeled the following basic attack strategies, each of which was parameterized with up to 2 parameters $p_1 \in [0, 1]$ and $p_2 \in \mathbb{N}$. The waveforms some of them are named after correspond to the evolution of the error rates:

RANDOM	For each interaction draw error rate randomly from a uniform distribution in $[0, p_1]$.
CONSTANT	Choose constant error rate of p_1.
DIRAC	Reiterate: Pretend to be a honest worker for a number of p_2 rounds, then return a response with error rate p_1.
SINE	For the ith interaction, choose error rate $p_1 \cdot (0.5 + 0.5 \cdot \sin_{p_2}(i))$, where p_2 parameterizes the frequency of the sine function. The constants 0.5 normalize the sine function into $[0, 1]$.

TABLE IV
PROPORTION OF ATTACK STRATEGIES AMONG BEST RESPONSES

RANDOM	CONSTANT	DIRAC	SINE	SQUARE	SAW
0.086	0.143	0.429	0.171	0.086	0.086

SQUARE Similarly to SINE, choose error rate $p_1 \cdot (0.5 + 0.5 \cdot \text{sgn}(\sin_{p_2}(i)))$.

SAW For the ith interaction, choose error rate $p_1 \cdot (\frac{i \bmod p_2}{p_2})$.

Note that the CONSTANT strategy also describes the behavior of honest workers which do return errors with a very low error rate p_1. We modeled the strategy ($s \in \{1, \ldots, 6\}$) itself as another parameter that the attacker had to choose. So the genetic algorithm optimized the pair (s, p_1, p_2) in order to maximize the attacker's utility against a given trust model configuration.

C. Experiments and Results

For the experiments, we set the tolerable error rate to $t_e = 0.1$, which could be adapted if the model was optimized for a specific application area. To determine a sufficiently significant mean utility, each game consisted of 2^{10} rounds, and was played 2^9 times. For the genetic algorithm we found that for 2^7 individuals, a mutation rate of 0.1, a selection pressure of $\mathcal{S} = 15$ (we used no explicit elitism though) and a convergence criterion of 0.1, the algorithm performed well. If several best responses to the same trust model configuration are found, the one with the worst utility for the master is chosen (see Def. 1). In conformance with [15] we considered an error rate of 0.0022 for the honest workers. We choose a fraction of 0.1 of malicious workers.

In our experiments we found that the DIRAC attack strategy was among all considered strategies by far the most effective one, and after several generations, other strategies often "died out". Table IV shows the percentage of the different strategies among the best responses.

TABLE V
THE MASTER'S UTILITIES WHEN ASSUMING BEST RESPONSES OF THE ATTACKER (THE MAXIMUM UTILITY IS FRAMED).

s_T		$\mathcal{U}_T(s_T, \text{br}^*(s_T))$
A	B	
0.1	1.0	80.9
⋮	⋮	⋮
0.4	21.0	83.0
⋮	⋮	⋮
0.9	26.0	81.7

Table V shows a small excerpt of the utilities for the master against the best responses of malicious workers. Also considering the utilities not shown in the table, a peak is reached at $(A, B) = (21.0, 0.4)$. This is also illustrated in Fig. 2 and Fig. 3, which show the impact of parameters A and B on the utility $\mathcal{U}_T$. Figure 2 shows the utilities for varying A, where each line corresponds to one value of B smaller than 14. Clearly, for any B a peak in the utility is at $A = 0.4$. For higher B, the impact of A became weaker. Figure 3 shows the impact of B on the utility, where for each B several values of A are considered. The figure reveals that the utilities $\mathcal{U}_T$ increase for increasing B up to 21 and then drop off. For higher values of B, the results appeared to become more and more random. This can be explained by the fact that for higher B, the selection of the workers depends less on their trust values and so, workers are more randomly chosen. As a consequence, the variance in the payoffs of the trustor gets bigger. Hence, too high values of B are not desirable since they make the performance of the trust model hardly predictable. To counter this, one could linearly decrease the value of B over time, as proposed in [13]. Then, the initial value and the gradient would be the parameters to be optimized.

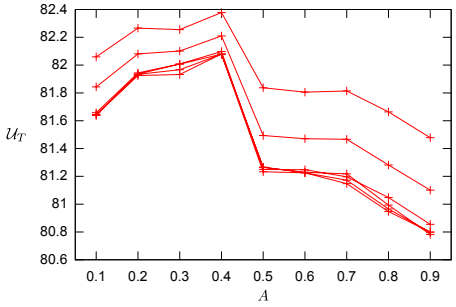

Fig. 2. Trustor Utility $\mathcal{U}_T$ for varying A (and different $B < 14$).

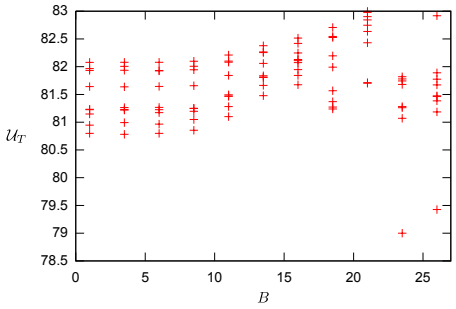

Fig. 3. Trustor Utility $\mathcal{U}_T$ for varying B (and $0.1 \leq A \leq 0.9$).

V. DISCUSSION

The strategies of the trustor are concrete trust model configurations (see Sect. II-A). Thus, our solution concept yields an optimized but fixed trust model configuration. In game-theoretic terms this means that the trustor basically plays a *pure strategy*. We believe that to compute the payoff for a *mixed strategy*, i.e. a strategy where the trustor uses different configurations with certain probabilities, would hardly be feasible in practice. The reason for this would be the complex dynamics in a game, e.g. caused by the honest players and the dynamic adaptation of the attackers to the changes in the trust model configuration. This raises the question of whether game-theory is the right approach for handling this complexity. We rather suggest that a configuration determined by a procedure

like ours serves as a starting configuration, and the trust model then adapts according to feedback it gets from its environment.

Instead of genetic algorithms, other search techniques could be used to search the attack strategy space. We have chosen a genetic algorithm, because it can flexibly be adapted to very different search spaces. Actually, depending on the attack strategy space, the fitness function can be nonlinear or even partially non-continuous – if this is taken to extremes, even a genetic algorithm will struggle. Furthermore, the structure of genetic algorithms makes them good candidates for parallelization, and fault tolerance seems to be inherent [16]. However, there are also reasons against genetic algorithms in our case. First, the configuration of the genetic algorithm itself can be tricky. In our scenario (Sect. IV), the results lead us to believe that we had succeeded in tuning the genetic algorithm sufficiently. Nevertheless, genetic algorithms do not guarantee convergence to global optima. Finally, the performance of our genetic algorithm suffers from the expensive computation of the fitness function. This is due to the fact that a game has to be repeated often enough to make the results statistically significant. Hence, a parallel execution is advisable.

VI. Related Work

A. Evidence-Based Trust Models

1) Parameter Usage: Many existing evidence-based trust models are parameterizable. Buchegger and Le Boudec [17] use several aging factors and give a rule of thumb for how to choose them. Wang and Vassileva use *learning rates* [18] which correspond to the aging factor in our model. Capra and Musolesi [19] use aging factor-like parameters that have to be defined subjectively. In the FIRE model [20], a "recency scaling factor" (an aging factor) and the "temperature" (parameter B in our model) are used. Also other models [21]–[24] use tunable aging factors, and [25] use an *aging function*. In [26], a modifiable threshold determines when an individual can be considered as trustworthy. Parameters for balancing trust and reputation values are used in [27]–[29]. How to systematically tune all these parameters has to the knowledge of the authors not yet been addressed.

2) The ART-testbed: In [30], the *ART testbed* was introduced to enable a comparison of the various trust- and reputation models proposed in the literature. In this testbed, several agents, of which each employs a certain trust model, compete in a turn-based game. The performance of an agent is measured both from an agent- and a system-perspective according to certain objective criteria. Still, the testbed itself does not allow for optimizing the parameters of trust models.

3) The Beta Distribution for Trust: The way our trust model computes the trust and uncertainty values is based on the mean and the variance of the Beta distribution. Also other evidence-based trust models use the beta distribution to this end ([17], [21], [23], [31]–[35]). However, the way how the experiences are processed, and how trust and uncertainty values are used for decision-making, differs from model to model.

B. Result Verification

Sarmenta [12] proposed to combine redundancy with *spot-checking*, where workers are assigned computations, for which the correct result is already known, with a certain rate. The author estimates the credibility of workers in order to reduce the overhead of the verification process. Workers that get caught returning incorrect results are blacklisted and their results are ignored (if identities can be checked). In their model, attackers are described by a Bernoulli process and return correct and incorrect results with a specific probability. Assuming this attacker model, the author attains probabilistically guaranteed levels of correctness. However, their model does not account for attackers that follow different strategies, as for instance the strategies described in Section IV-B2. Also, their model can cause honest workers to accidentally get blacklisted, which might increase the costs in the long run.

The framework in which our trust model operates ([9]) is closely related to "Quiz" [36], a scheme for result verification in peer-to-peer grids. In their scheme, a master delegates a whole package of computations to a worker. In such a package, quizzes are interspersed for which the correct result is known (as in our case). If one of the quizzes is answered incorrectly, the whole package is discarded and the trustworthiness of the corresponding worker is reduced (by the use of some trust model) and it is possibly blacklisted. Here, our model is fundamentally different, since it is designed for scenarios where a certain error rate t_{e} is tolerable. As a consequence, their trust models assume each incorrect result as indication for untrustworthiness, which is not the case in our model. Also, the trust models they use do not account for uncertainty.

Germain-Renaud and Monnier-Ragaigne [37] proposed a scheme for result verification in which they sequentially test a set of work units (they "call an oracle" to this end), and accept/reject as soon as it is statistically reasonable. Their approach requires that the oracle verifies the work units at runtime, whereas our model is applicable for cases where no such oracle is available and prepared challenges can be used.

VII. Conclusion and Future Work

In this paper, we presented an approach for systematically tuning parameters of evidence-based trust models. We first use a genetic algorithm to search for an attacker's best responses against different trust model configurations. By then choosing the configuration with the highest utility against the corresponding best response, one finds a Nash equilibrium.

We hope that our work contributes to the development of a general *benchmark* for evidence-based trust models. There exist approaches that test competing trust models in specific scenarios (e.g. [30], [38]). However, these can for instance not be used to benchmark the trust model optimized in our paper. A generic benchmark could for example optimize the trust models against sets of attack strategies (as shown in this paper), and then compare the expected utilities.

Our work gives a validated starting configuration for trust models. The problem of how to adjust parameters at runtime in order to react to the environment is a subject for future work.

Also, in the tuning procedure that we explored, we fixed the ratio of attackers and had all of them using the same strategy in a game. We plan to extend this and consider different fractions of attackers, and attackers that use diverse strategies. Finally, we assume that attackers act perfectly rational, i.e. they know their best response and choose it. Here, more elaborate game-theoretic solution concepts that also account for the case that opponents choose suboptimal strategies would be helpful (e.g., see [39]).

REFERENCES

[1] D. Artz and Y. Gil, "A survey of trust in computer science and the semantic web," *Web Semant.*, vol. 5, no. 2, pp. 58–71, 2007.

[2] S. D. Ramchurn, T. D. Huynh, and N. R. Jennings, "Trust in multi-agent systems," *Knowl. Eng. Rev.*, vol. 19, no. 1, pp. 1–25, 2004.

[3] J. Sabater and C. Sierra, "Review on computational trust and reputation models," *Artif. Intell. Rev.*, vol. 24, no. 1, pp. 33–60, 2005.

[4] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: an experiment in public-resource computing," *Commun. ACM*, vol. 45, no. 11, pp. 56–61, 2002.

[5] D. Fudenberg and J. Tirole, *Game theory*. Cambridge, MA: MIT Press, 1991.

[6] J. Nash, "Noncooperative games," *Annals of Mathematics*, vol. 14, no. 2, pp. 124–143, 1951.

[7] J. H. Holland, *Adaptation in natural and artificial systems*. Cambridge, MA, USA: MIT Press, 1992.

[8] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, January 1989.

[9] E. Staab, V. Fusenig, and T. Engel, "Towards trust-based acquisition of unverifiable information," in *Cooperative Information Agents XII*, ser. LNCS (LNAI), M. Klusch, M. Pechoucek, and A. Polleres, Eds., vol. 5180. Springer Verlag, September 2008, pp. 41–54.

[10] P. Golle and S. G. Stubblebine, "Secure distributed computing in a commercial environment," in *Proc. of the 5th Int. Conf. on Financial Cryptography (FC '01)*. Springer Verlag, 2002, pp. 289–304.

[11] D. Kondo, M. Taufer, C. L. Brooks, H. Casanova, and A. A. Chien, "Characterizing and evaluating desktop grids: An empirical study," in *Proc. of the Int. Parallel and Distributed Processing Symposium (IPDPS'04)*. IEEE Computer Society, 2004, pp. 26–35.

[12] L. F. G. Sarmenta, "Sabotage-tolerance mechanisms for volunteer computing systems," in *Proc. of the 1st Int. Symposium on Cluster Computing and the Grid (CCGrid '01)*. IEEE Computer Society, 2001, pp. 337–346.

[13] W. B. Powell, *Approximate Dynamic Programming*, 1st ed. John Wiley & Sons, Inc., September 2007.

[14] K. Hoffman, D. Zage, and C. Nita-Rotaru, "A survey of attack and defense techniques for reputation systems," Purdue University, Tech. Rep. CSD TR #07-013, 2007.

[15] D. Kondo, F. Araujo, P. Malecot, P. Domingues, L. M. Silva, G. Fedak, and F. Cappello, "Characterizing result errors in internet desktop grids," in *Proc. of the 13th European Conf. on Parallel Processing (EURO-PAR '07)*, ser. LNCS, vol. 4641. Springer Verlag, 2007, pp. 361–371.

[16] D. L. González, F. F. de Vega, and H. Casanova, "Characterizing fault tolerance in genetic programming," in *Proc. of the 2009 Workshop on Bio-Inspired Algorithms for Distributed Systems (BADS '09)*, 2009, pp. 1–10.

[17] S. Buchegger and J.-Y. L. Boudec, "A robust reputation system for mobile ad hoc networks," EPFL-IC-LCA, CH-1015 Lausanne, Tech. Rep. IC/2003/50, July 2003.

[18] Y. Wang and J. Vassileva, "Trust and reputation model in peer-to-peer networks," in *Proc. of the 3rd Int. Conf. on Peer-to-Peer Computing (P2P '03)*. IEEE Computer Society, 2003, p. 150.

[19] L. Capra and M. Musolesi, "Autonomic trust prediction for pervasive systems," in *AINA '06: Proc. of the 20th Int. Conf. on Advanced Information Networking and Applications (Vol. 2)*. IEEE Computer Society, 2006, pp. 481–488.

[20] T. D. Huynh, N. R. Jennings, and N. R. Shadbolt, "An integrated trust and reputation model for open multi-agent systems," *Auton. Agents Multi-Agent Syst.*, vol. 13, no. 2, pp. 119–154, 2006.

[21] A. Whitby, A. Jøsang, and J. Indulska, "Filtering out unfair ratings in Bayesian reputation systems," in *Proc. of the 7th Int. Workshop on Trust in Agent Societies (at AAMAS '04)*, 2004.

[22] M. Kinateder, E. Baschny, and K. Rothermel, "Towards a generic trust model - comparison of various trust update algorithms," in *iTrust '05: Proc. of the 3rd Int. Conf. on Trust Management*. Springer Verlag, 2005, pp. 177–192.

[23] T. B. Klos and H. L. Poutré, "Decentralized reputation-based trust for assessing agent reliability under aggregate feedback," in *Proc. of the 17th Belgium-Netherlands Conf. on Artificial Intelligence (BNAIC '05)*, 2005, pp. 357–358.

[24] L. Xiong and L. Liu, "A reputation-based trust model for peer-to-peer ecommerce communities," in *Proc. of the 5th Int. Conf. on Electronic Commerce (EC '03)*. ACM, 2003, pp. 228–229.

[25] M. Ion, A. Danzi, H. Koshutanski, and L. Telesca, "A peer-to-peer multidimensional trust model for digital ecosystems," in *Proc. of the 2nd IEEE International Conference on Digital Ecosystems and Technologies*, February 2008, pp. 461–469.

[26] C. Zouridaki, B. L. Mark, M. Hejmo, and R. K. Thomas, "Robust cooperative trust establishment for MANETs," in *Proc. of the 4th ACM Workshop on Security of ad hoc and Sensor Networks (SASN 2006)*. ACM, October 2006, pp. 23–34.

[27] M. Virendra, M. Jadliwala, M. Chandrasekaran, and S. Upadhyaya, "Quantifying trust in mobile ad-hoc networks," in *Proc. of the IEEE International Conference on Integration of Knowledge Intensive Multi-agent Systems (KIMAS'05)*. IEEE Computer Society, 2005, pp. 65–70.

[28] P. B. Velloso, R. P. Laufer, O. C. M. B. Duarte, and G. Pujolle, "A trust model robust to slander attacks in ad hoc networks," in *Proc. of the Workshop in Advanced Networking and Communications (ANC), at ICCCN'08*, August 2008.

[29] V. Balakrishnan, V. Varadharajan, P. Lucs, and U. K. Tupakula, "Trust enhanced secure mobile ad-hoc network routing," in *Workshop Proceedings Vol. 2 of the 21st Int. Conf. on Advanced Information Networking and Applications (AINA'07)*. IEEE Computer Society, 2007, pp. 27–33.

[30] K. Fullam, T. B. Klos, G. Muller, J. Sabater, A. Schlosser, Z. Topol, K. S. Barber, J. S. Rosenschein, L. Vercouter, and M. Voss, "A specification of the agent reputation and trust (ART) testbed: experimentation and competition for trust in agent societies," in *AAMAS '05: 4th Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, 2005, pp. 512–518.

[31] L. Mui, M. Mohtashemi, and A. Halberstadt, "A computational model of trust and reputation for e-businesses," in *Proc. of the 35th Annual Hawaii Int. Conf. on System Sciences (HICSS '02)*. IEEE Computer Society, 2002, pp. 188–196.

[32] Y. Wang and M. P. Singh, "Formal trust model for multiagent systems," in *IJCAI '07: Proc. of the 20th Int. Joint Conf. on Artificial Intelligence*, 2007, pp. 1551–1556.

[33] W. T. L. Teacy, J. Patel, N. R. Jennings, and M. Luck, "Travos: Trust and reputation in the context of inaccurate information sources," *Auton. Agents Multi-Agent Syst.*, vol. 12, no. 2, pp. 183–198, 2006.

[34] A. Jøsang and R. Ismail, "The beta reputation system," in *Proc. of the 15th Bled Conf. on Electronic Commerce*, 2002, pp. 324–337.

[35] G. H. Nguyen, P. Chatalic, and M.-C. Rousset, "A probabilistic trust model for semantic peer-to-peer systems," in *Proc. of the 18th Eur. Conf. on Artificial Intelligence (ECAI '08)*. IOS Press, 2008, pp. 881–882.

[36] S. Zhao, V. Lo, and C. GauthierDickey, "Result verification and trust-based scheduling in peer-to-peer grids," in *Proc. of the 5th IEEE Int. Conf. on Peer-to-Peer Computing (P2P '05)*. IEEE Computer Society, 2005, pp. 31–38.

[37] C. Germain-Renaud and D. Monnier-Ragaigne, "Grid result checking," in *Proc. of the 2nd Conf. on Computing Frontiers (CF '05)*. ACM, 2005, pp. 87–96.

[38] "Trading agent competition (TAC)," http://www.sics.se/tac/, 2009.

[39] J. Pita, M. Jain, F. Ordóñez, M. Tambe, S. Kraus, and R. Magori-Cohen, "Effective solutions for real-world Stackelberg games: When agents must deal with human uncertainties," in *Proc. of the 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS '09)*. IFAAMAS, May 2009, pp. 369–376.