Supplementary material for ARMS: Automated rules management system for fraud detection

Supplementary Algorithms

Algorithm S1 Blacklist propagation.

```
function computeBlacklistDependencies($\mathbf{R}$, $\mathbf{X}$, $\mathcal{X}$, $\mathcal{B}$)
    $\mathbf{BL} \leftarrow \{\}$
    $\mathbf{BD} \leftarrow \{\}$
    for all $\mathbf{x} \in \mathbf{X}$ do
        for all $R_j \in \mathcal{B}^u$ do
            if $r_j \neq -1$ then
                for all $X_l \in \mathcal{X}$ that $R_j$ blacklists do
                    $\mathbf{BL}[(R_j, X_l : x_l)]$.append($[\mathbf{x}.time, +\infty]$)
                    for all $R_q \in \mathcal{B}^c$ that checks $X_l$ do
                        $\mathbf{BD}[\mathbf{x}]$.add($R_j \prec R_q$)
                    end for
                end for
            end if
            if $r_j = -1$ then
                for all $X_l \in \mathcal{X}$ that $R_j$ can blacklist do
                    if $\mathbf{x}.time$ is in any $\mathbf{BL}[(R_j, X_l : x_l)]$ then
                        $r_j \leftarrow p_j$
                    end if
                end for
            end if
        end for
        for all $\{x_l \in \mathbf{x} \mid x_l$ is in any active blacklist $\}$ do
            if $(\nexists R_q \in \mathcal{B}^c \mid p_q \neq -1)$ then
                for all $\{R_j \in \mathcal{B}^u \mid (R_j, X_l : x_l) \in \mathbf{BL}\}$ do
                    $\mathbf{BL}[(R_j, X_l : x_l)]$.last() $\leftarrow [\_, \mathbf{x}.time]$
                end for
            end if
        end for
        for all $R_q \in \mathcal{B}^c$ do
            if $r_q \neq -1$ and $|\{(R_i \prec R_q) \in \mathbf{BD}[\mathbf{x}] \mid R_i \in \mathcal{B}^u\}| = 0$ then
                $\mathbf{BD}[\mathbf{x}]$.add($R_q \prec R_q$))
            end if
        end for
    end for
    return $\mathbf{BD}$
end function
```

Before optimization, ARMS computes blacklist dependencies (Supplementary Algorithm 1) to measure the performance of blacklisting rules properly. It creates two empty dictionaries **BL** and **BD** (lines 2 and 3):

- **BL** (for *blacklist*) stores all the entities x_l (e.g., username, e-mail, device) blacklisted by any given rule R_j for a feature $\mathcal{X}_l$. Entries save an interval $[t_0, t_1]$ corresponding to the timestamps of the start and the end of the period in which the entity was blacklisted (i.e., $\mathbf{BL}[R_j, \mathcal{X}_l : x_l] = [t_0, t_1]$). Blacklisting can be done by rules or human operators (e.g., at t_0), whereas the removal of entities from the blacklist is only done by human operators (e.g., at t_1).

- **BD** (for *blacklist dependencies*) stores, for each transaction, the dependencies between rules. Hence, $\mathbf{BD}[\mathbf{x}]$ contains all dependencies between blacklist checker rules $R_q \in \mathcal{B}^c$ and (a subset of) blacklisting rules $\mathcal{R}^u \subseteq \mathcal{B}^u$, represented as $\mathcal{R}^u \prec R_q$. We ensure that if we turn off all blacklist updater rules triggered by the transaction, we turn off the blacklist checker rules that depend on them. Different blacklist checker rules check different entities.

ARMS traverses all transactions $\mathbf{x} \in \mathbf{X}$, ordered by time, to compute the dependencies (lines 4-21):

1. At first, ARMS updates **BL** and **BD** depending on the triggers of each blacklisting rule $R_j \in \mathcal{B}^u$ (lines 5-14). If rule R_j triggered for transaction $\mathbf{x}$ (line 6), ARMS adds all blacklisted entities (e.g., email, card, device) to **BL** for the period from the transaction's timestamp $\mathbf{x}.time$ until the entity is, potentially, removed from the blacklist (lines 7-8). Then, for all blacklist checker rules $R_q \in \mathcal{B}^c$ that checks entity X_l, ARMS adds the dependency of R_q relative to R_j (lines 9-10). If rule R_j did not trigger for transaction $\mathbf{x}$, but the transaction contains an entity $X_l : x_l$ that was blacklisted previously (lines 12-13), then ARMS turns the rule on *a posteriori* for evaluation purposes, i.e., $r_j \leftarrow p_j$ (line 14).

2. Second, ARMS checks if there are previously blacklisted entities that were removed from the blacklist (lines 15-18). If a transaction with a previously blacklisted entity $X_l : x_l$ did not trigger the rules that check it, we conclude that it was manually removed from the blacklist. ARMS does this by checking for each feature $x_l \in \mathbf{x}$ that is currently blacklisted (if any) (line 15) if it triggered any blacklist checker rule R_q (line 16). If it did not, we know that the entity was removed from the blacklist and update **BL** for each blacklisting rule R_j that had added $X_l : x_l$ to the blacklist (line 17) with the transaction's timestamp (line 18).

3. Third, ARMS checks if any blacklist checker rule $R_q \in \mathcal{B}^c$ depends solely on manual decisions and not on blacklisting rules (lines 19-21). Typically, if all blacklist updater rules that a blacklist checker rule depends on are inactive, then the blacklist checker rule is considered inactive. Nonetheless, if the blacklist checker rule only depends on itself (lines 20-21), then it stays on regardless of blacklist updater rules unless explicitly turned off.

Finally, ARMS returns the **BD** (line 22), to be considered when evaluating of the configurations found during optimization.

Algorithm S2 Random search optimization.

θ: { rule shutoff probability ρ, rule priority shuffle probability γ }

1: **function** RANDOM.OPTIMIZE($\mathbf{X}$, $\mathbf{R}$, $\boldsymbol{\ell}$, $\mathbf{p}$, a, $\mathbf{BD}$, λ, Ω^1, θ)
2: $\quad$ $\mathbf{p}^{\mathbf{best}} \leftarrow \mathbf{p}$
3: $\quad$ $\Omega^{best} \leftarrow \Omega^1$
4: $\quad$ **while** STOPPINGCRITERIANOTMET() **do**
5: $\quad\quad$ $\mathbf{p}^{\mathbf{rand}} \leftarrow \mathbf{p}$
6: $\quad\quad$ **for all** $p_i \in \mathbf{p}^{\mathbf{rand}}$ **do**
7: $\quad\quad\quad$ **with** $\gamma\%$ probability, **do**:
8: $\quad\quad\quad\quad$ $p_i \leftarrow$ RANDOMPRIORITYSHUFFLE(p_i, a)
9: $\quad\quad\quad$ **with** $\rho\%$ probability, **do**:
10: $\quad\quad\quad\quad$ $p_i \leftarrow -1$
11: $\quad\quad$ **end for**
12: $\quad\quad$ $\Omega^{rand} \leftarrow$ EVALUATE($\mathbf{X}$, $\mathbf{R}$, $\boldsymbol{\ell}$, $\mathbf{p}^{\mathbf{rand}}$, a, $\mathcal{B}$, $\mathbf{BD}$, λ)
13: $\quad\quad$ **if** $\Omega^{rand}_{loss} < \Omega^{best}_{loss}$ **then**
14: $\quad\quad\quad$ $\Omega^{best} \leftarrow \Omega^{rand}$
15: $\quad\quad\quad$ $\mathbf{p}^{\mathbf{best}} \leftarrow \mathbf{p}^{\mathbf{rand}}$
16: $\quad\quad$ **end if**
17: $\quad$ **end while**
18: $\quad$ **return** ($\mathbf{p}^{\mathbf{best}}, \Omega^{best}$)
19: **end function**

Initially, the best rule configuration, $\mathbf{p}^{\mathbf{best}}$, is the original system (lines 2-3). Then, until meeting the stopping criteria, ARMS generates random priority vectors $\mathbf{p}^{\mathbf{rand}}$, evaluates them, and saves the best one (lines 4-14):

1. First, ARMS initializes $\mathbf{p}^{\mathbf{rand}}$ to be $\mathbf{p}$ (line 5).
2. Second, for each rule, ARMS performs two operations to generate a new vector, $\mathbf{p}^{\mathbf{rand}}$, as (lines 6-10):
   (a) It changes the rule priority with probability γ (lines 7-8).
   (b) It turns the rule off with probability ρ (lines 9-10).
3. Third, it evaluates $\mathbf{p}^{\mathbf{rand}}$ as Ω^{rand} (line 11).
4. Finally, if the configuration, $\mathbf{p}^{\mathbf{rand}}$, is the best so far, the system saves it and Ω^{rand} as $\mathbf{p}^{\mathbf{best}}$ and Ω^{best}, respectively, and returns both (lines 12-14).

In the end, ARMS returns $\mathbf{p}^{\mathbf{best}}$ and Ω^{best} (line 15).

Algorithm S3 Greedy expansion optimization.

θ: { backtracking $bt \in \{true, false\}$ }

1: **function** GREEDY.OPTIMIZE($\mathbf{X}$, $\mathbf{R}$, $\boldsymbol{\ell}$, $\mathbf{p}$, a, $\mathbf{BD}$, λ, Ω^1, θ)
2: $\quad \mathbf{p^{best}} \leftarrow \mathbf{p}$
3: $\quad \Omega^{best} \leftarrow \Omega^1$
4: $\quad \mathbf{p^{keep}} \leftarrow (-1, ..., -1)$
5: $\quad \mathbf{p^{greedy}} \leftarrow (-1, ..., -1)$
6: $\quad Q \leftarrow \emptyset$
7: $\quad$ **while** $|Q| < |\mathcal{R}|$ **and** STOPPINGCRITERIANOTMET() **do**
8: $\qquad R_{keep} \leftarrow$ None
9: $\qquad \Omega^{keep} \leftarrow +\infty$
10: $\qquad$ **for all** $\{R_j \in \mathcal{R} \mid R_j \notin Q\}$ **do**
11: $\qquad\quad p_j^{greedy} \leftarrow p_j$
12: $\qquad\quad \Omega^{greedy} \leftarrow$ EVALUATE($\mathbf{X}$, $\mathbf{R}$, $\boldsymbol{\ell}$, $\mathbf{p^{greedy}}$, a, $\mathcal{B}$, $\mathbf{BD}$, λ)
13: $\qquad\quad$ **if** $\Omega_{loss}^{greedy} < \Omega_{loss}^{keep}$ **then**
14: $\qquad\qquad R_{keep} \leftarrow R_j$
15: $\qquad\qquad \Omega^{keep} \leftarrow \Omega^{greedy}$
16: $\qquad\qquad \mathbf{p^{keep}} \leftarrow \mathbf{p^{greedy}}$
17: $\qquad\quad$ **end if**
18: $\qquad\quad p_j^{greedy} \leftarrow -1$
19: $\qquad$ **end for**
20: $\qquad Q$.ADD(R_{keep})
21: $\qquad$ **if** $\Omega_{loss}^{keep} < \Omega_{loss}^{best}$ **then**
22: $\qquad\quad \Omega^{best} \leftarrow \Omega^{keep}$
23: $\qquad\quad \mathbf{p^{best}} \leftarrow \mathbf{p^{keep}}$
24: $\qquad$ **end if**
25: $\qquad$ **if** bt is $true$ **and** ISBACKTRACKINGTIME() **then**
26: $\qquad\quad$ run greedy contraction to remove l rules, $l < |Q|$
27: $\qquad$ **end if**
28: $\quad$ **end while**
29: $\quad$ **return** $(\mathbf{p^{best}}, \Omega^{best})$
30: **end function**

The algorithm starts from the original system (lines 2-3) but all rules as *inactive* (lines 4-6). ARMS stores three different rules priority vectors:

- $\mathbf{p^{best}}$ stores the best rules configuration.
- $\mathbf{p^{keep}}$ stores the best rules configuration at each expansion.
- $\mathbf{p^{greedy}}$ is a temporary vector with all possible expansions.

Then, until either there are no more possible expansions or when the stopping criteria is met, ARMS evaluates each possible expansion, chooses the best one at each step, and saves the overall best, which is not necessarily the last expansion (lines 7-23):

1. First, ARMS evaluates what is the best possible rule addition to the current $\mathbf{p^{keep}}$ (lines 8-18), by looping over all combinations with one new rule added.
2. Second, ARMS checks if the current expansion, $\mathbf{p^{keep}}$, should replace the best configuration so far, $\mathbf{p^{best}}$ (lines 19-21). It it does, ARMS updates p^{best} and Ω^{best}.
3. Third, from time to time (e.g., after n iterations), ARMS tries to remove $[1, l]$ rules and checks if it improves results (lines 22-23). This is a mechanism to protect the system against redundant or detrimental expansions. We do not show pseudo-code for this part since greedy contraction is identical to greedy expansion, but removing instead of adding rules.

Finally, ARMS returns both $\mathbf{p^{best}}$ and Ω^{best} (line 24).

Algorithm S4 Genetic programming optimization.

θ: { Population size ψ, survivors fraction α, mutation probability ρ }

1: **function** GENETIC.OPTIMIZE($\mathbf{X}$, $\mathbf{R}$, $\boldsymbol{\ell}$, $\mathbf{p}$, a, $\mathbf{BD}$, λ, Ω^1, θ)
2: $\quad$ $\mathbf{p^{best}} \leftarrow \mathbf{p}$
3: $\quad$ $\Omega^{best} \leftarrow \Omega^1$
4: $\quad$ $\mathbf{P} \leftarrow$ GENERATEINITIALPOPULATION($\mathbf{R}$, $\mathbf{p}$, ψ, ρ)
5: $\quad$ **while** STOPPINGCRITERIANOTMET() **do**
6: $\quad\quad$ $(\mathbf{P}^{=}, \mathbf{P}^{-}) \leftarrow$ EVALUATEPOPULATION($\mathbf{P}$, α)
7: $\quad\quad$ $\mathbf{P}^{+} \leftarrow$ MUTATEANDCROSSOVER($\mathbf{P}^{=}$, α, ψ, ρ)
8: $\quad\quad$ $\mathbf{P} \leftarrow \{\mathbf{P}^{=}, \mathbf{P}^{+}\}$
9: $\quad$ **end while**
10: $\quad$ $(\mathbf{P}^{=}, \mathbf{P}^{-}) \leftarrow$ EVALUATEPOPULATION($\mathbf{P}$)
11: $\quad$ $\mathbf{p^{best}} \leftarrow \mathbf{P}^{=}_{\mathbf{1}}$
12: $\quad$ $\Omega^{best} \leftarrow$ EVALUATE($\mathbf{X}$, $\mathbf{R}$, $\boldsymbol{\ell}$, $\mathbf{p^{best}}$, a, $\mathcal{B}$, $\mathbf{BD}$, λ)
13: $\quad$ **return** $(\mathbf{p^{best}}, \Omega^{best})$
14: **end function**

15: **function** GENERATEINITIALPOPULATION($\mathbf{R}$, $\mathbf{p}$, ψ, ρ)
16: $\quad$ $\mathbf{P} \leftarrow \emptyset$
17: $\quad$ **for** $i \in [0, \psi[$ **do**
18: $\quad\quad$ $\mathbf{p}' \leftarrow \mathbf{p}$
19: $\quad\quad$ **for all** $p'_j \in \mathbf{p}'$ **do**
20: $\quad\quad\quad$ **with** ρ% probability, **do**:
21: $\quad\quad\quad\quad$ $p'_j \leftarrow -1$
22: $\quad\quad$ **end for**
23: $\quad\quad$ $\mathbf{P}[\mathbf{i}] \leftarrow \mathbf{p}'$
24: $\quad$ **end for**
25: $\quad$ **return P**
26: **end function**

27: **function** MUTATEANDCROSSOVER($\mathbf{P}^{=}$, α, ψ, ρ)
28: $\quad$ $\mathbf{P}^{+} \leftarrow \emptyset$
29: $\quad$ **for** $i \in [0, (1-\alpha) * \psi[$ **do**
30: $\quad\quad$ $\mathbf{p^{mother}} \leftarrow$ GETRANDOMVECTOR($\mathbf{P}^{=}$)
31: $\quad\quad$ $\mathbf{p^{father}} \leftarrow$ GETRANDOMVECTOR($\mathbf{P}^{=}$)
32: $\quad\quad$ $\mathbf{p^{child}} \leftarrow \mathbf{p^{mother}}$
33: $\quad\quad$ **for all** $p_j^{child} \in \mathbf{p^{child}}$ **do**
34: $\quad\quad\quad$ **with** 50% probability, **do**:
35: $\quad\quad\quad\quad$ $p_j^{child} \leftarrow p_j^{father}$
36: $\quad\quad$ **end for**
37: $\quad\quad$ **for all** $p_j^{child} \in \mathbf{p^{child}}$ **do**
38: $\quad\quad\quad$ **with** ρ% probability, **do**:
39: $\quad\quad\quad\quad$ $p_j^{child} \leftarrow$ RANDOMPRIORITYSHUFFLE(p_i, a)
40: $\quad\quad$ **end for**
41: $\quad\quad$ $\mathbf{P}^{+}$.ADD($\mathbf{p^{child}}$)
42: $\quad$ **end for**
43: $\quad$ **return** $\mathbf{P}^{+}$
44: **end function**

We initiate $\mathbf{p^{best}}$ to be the original system (lines 2-3). Before optimization, ARMS initializes the random population (line 4, lines 15-26) . Then, until meeting the stopping criteria, ARMS continuously improves the configurations and saves the best ones (lines 5-11).

1. First, ARMS evaluates the current population, $\mathbf{P}$, of configurations: the α% best, $\mathbf{P}^{=}$, survive for the next iteration, while the others, $\mathbf{P}^{-}$, are discarded (line 6). It is the *selection step*, typical in genetic algorithms.

2. Second, ARMS generates new rule configurations (line 7), $\mathbf{P}^{+}$, to replace the discarded ones, using *crossover* between survivors and random genetic mutations (lines 27-44). The pool of inheritance comprises all surviving rule configurations $\mathbf{P}^{=}$ and $(1-\alpha) * \psi$ new rule configurations, $\mathbf{P}^{+}$ (lines 29-41), by repeating:

(a) A *mother*, $\mathbf{p^{mother}}$, and a *father*, $\mathbf{p^{father}}$, configurations are randomly sampled from $\mathbf{P}^{=}$ (lines 30-31).

(b) The *child*, $\mathbf{p^{child}}$, initially inherits the complete rule configuration of the mother, $\mathbf{p^{mother}}$ (line 32).

(c) Then, with 50% probability, it swaps each *gene* with the gene of its father, $\mathbf{p^{father}}$ (lines 33-35).

(d) It performs random rule priority shuffling with probability ρ (lines 37-40).

(e) It adds the new rule configuration, $\mathbf{p^{child}}$, to $\mathbf{P}^{+}$ (line 41).

3. Third, ARMS augments $\mathbf{P}^{=}$ with $\mathbf{P}^{+}$ and the selection process repeats (line 8). Note that the best α rule configurations remain unmodified, thus it is guaranteed that the best solution found by ARMS is in $\mathbf{P}$ at the end of the process.

4. Fourth, when meeting the stopping criteria, ARMS evaluates the final population (line 10) and saves the best configuration, $\mathbf{p^{best}}$, (line 11) as well as its performance, Ω^{best} (line 12).

As above, ARMS returns $\mathbf{p^{best}}$ and Ω^{best} (line 13).