engineering proceedings

MDPI

Proceeding Paper

Impact of Dependency Density on Firewall Rule Ordering Optimization Using Discrete Cuckoo Search †

Fadwa Bezzazi and Dounia Lotfi *

LRIT, Faculty of Sciences, Mohammed V University in Rabat, Rabat P.O. Box 1014 RP, Morocco; fadwa.bezzazi@um5r.ac.ma

* Correspondence: d.lotfi@um5r.ac.ma

† Presented at the 8th edition of the International Conference on Advanced Technologies for Humanity (ICATH 2026), Rabat, Morocco, 15–17 July 2026.

Abstract

Firewall rule ordering is known to be an NP-hard problem in network security. It aims to reduce packet processing cost by placing frequently matched rules earlier in the policy while preserving semantic correctness under inter-rule dependencies. Although metaheuristic methods have shown promising results for this task, the effect of dependency structure on optimization quality remain poorly studied. In this paper we extend our previous Reordering Discrete Cuckoo Search (RDCS) method by introducing a dependency matrix to represent precedence constraints explicitly and a freedom rate metric to quantify the reorderable part of the search space. Experiments were conducted on synthetic firewall policies of 100 rules under a density levels. The results show that the dependency matrix provides a clear optimization benefit, with gains that increase as the number of constraints grows. They also show that RDCS consistently achieves the lowest final fitness compared with random Search, Genetic Algorithm, and Simulated Annealing under the same evaluation conditions. In addition, while dependency density affects the absolute quality of the solutions, the optimization capability of RDCS remains stable. These findings highlight the practical importance of dependency-aware optimization in firewall rule management.

Keywords: firewall optimization; rule reordering; Discrete Cuckoo Search; dependency constraints; network security

1. Introduction

Firewall rule management is an important issue in network administration because security policies are becoming larger and more complex. In a firewall, packets are checked sequentially against an ordered list of rules. The action of the first matching rule is applied [1]. Therefore, rule order directly affects performance. When frequently matched rules are placed earlier, the average number of comparisons is reduced. This improves packet processing efficiency [2,3]. However, firewall rules cannot always be reordered freely. Some rules are semantically dependent. Changing their order may modify the filtering decision for some packets and may violate the intended policy behavior [2]. These dependencies create precedence constraints between rules. As the number of dependencies increases, the set of valid orderings becomes smaller. Consequently, the feasible search space is reduced and the optimization problem becomes more difficult. Optimizing firewall rule order under such constraints is a difficult combinatorial problem related to precedence-constrained scheduling and known to be NP-hard [4]. Previous work has mainly focused on anomaly detection and resolution [5,6], packet classification techniques [7], or heuristic

Check for updates

Academic Editors: Brahim El Bhiri, Ayman S. Mosallam and Younes Karfa Bekali

Published: 20 November 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

approaches for firewall optimization [8]. On the other hand, the effect of dependency density on optimization quality has received limited attention. In our previous work [9], we introduced the Reordering Discrete Cuckoo Search (RDCS) algorithm, inspired by Cuckoo Search [10], and showed that it can achieve competitive results for firewall rule ordering. However, the role of inter-rule dependencies was not studied explicitly, an the influence of dependency density on both solution quality and search space size remained unclear. This paper addresses that gap through three main contributions. In the beginning we introduce a formal dependency matrix to represent precedence constraints between rules explicitly. After that we define the freedom rate, a metric that quantifies the proportion of rule pairs that remain reordeable [11]. Finally, we compare RDCS with constrained random search and a constrained greedy heuristic under different dependency density levels using synthetic rule sets and a zipf traffic model. The results provide a clearer view of how structural constraints influence firewall rule ordering performance. The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 presents the problem formulation, including the dependency matrix and the freedom rate. Section 4 describes the Reordering Discrete Cuckoo Search. Section 5 reports the experimental setup and results and discusses the findings, and Section 6 concludes the paper.

2. Related Work

Firewall rule ordering has benn widely studied as an important optimization problem in network security. The main objective is to reduce packet matching cost while preserving the semantic correctness of the policy. One of the earliest formal studies was proposed by Fulp [2], who modeled firewall rule as a directed acyclic graph in which edges represent precedence constraints between rules. This work showed that the problem is related to precedence-constrained scheduling and is computationally hard. It also introduced a heuristic approach for obtaining near-optimal rule orderings. Later, Elalfy [8] formulated the problem as a binary integer programming task and proposed a genetic algorithm to improve rule ordering under Zipf-distributed traffic. More recently, Fonseca et al. [12] presented a two-stage approach combining topological sorting and genetic search to build valid and efficient orderings. Although these studies confirmed the importance of optimization in firewall rule management, the did not explicitly analyze how the density of dependency constraints affects optimization quality. More generally, metaheuristic algorithms have been widely used for hard combinatorial optimization problems, including constrained sequencing and scheduling. Genetic Algorithms [13] have often been applied to ordering problems, but they usually require problem-specific operators and repair mechanisms to preserve valid solutions under precedence constraints. Simulated Annealing [14] has also been used as a search strategy for permutation problems, but its efficiency strongly depends on the choice of neighborhood structure and cooling schedule. In contrast, Cuckoo Search, introduced by yan and deb [10], has attracted attention because of its simple design and its balance between exploration and exploitation. Its search mechanism, based on Levy flight, make it a suitable candidate for discrete optimization problem, including constrained ordering tasks [15]. Another important line os research concerns the modeling of dependencies between firewall rules. Abedin et al. [6] studies anomaly detection and resolution in firewall policies and highlighted several forms of interaction between rules, such as shadowing, correlation and redundancy. Similarly, Al-shaer and Hamed [5] analyzed policy anomalies in distributed firewalls and provided a formal view of rule conflicts and dependencies. These studies are important because they show that rule interactions must be handled carefully to preserve policy behavior. However, their focus was mainly on correctness and anomaly resolution rather than on performance optimization. Based on these previous works, it appears that two aspects have often been studied separately:

dependency modeling on one side, and metaheuristic optimization on the other. In contrast, our work combines both aspects by explicitly representing inter-rule dependencies through a dependency matrix and by studying their impact on optimization performance. The relationship between dependency density and the quality of firewall rule ordering remains a challenging task. This is the gap addressed in this paper through the use of the freedom rate and the comparative evaluation of constrained optimization methods.

3. Problem Formulation

In firewall rule ordering, the goal is to reduce packet matching cost while preserving the semantic correctness of the policy. This section introduces the rule ordering problem, the dependency matrix, the freedom rate, and the resulting constrained optimization model.

3.1. Firewall Rule Ordering Problem

Let's consider $R = \{r_1, r_2, \dots, r_n\}$ a set of n firewall rules. Each rule r_i is defined by a matching condition on packet header fields and an action, such as accept or deny. A firewall policy is an ordered list of rules evaluated sequentially under the first-match principle: for any packet p, the action of the first matching rule is applied. Let f_i be the weight of an active rule r_i, i.e., the probability that a packet matches r_i before any other rule. In this work, rule frequencies follow Zipf's law, which models skewed traffic distributions [16]. The frequency of the rule ranked at position k is given by:

$$f(k) = \frac{1/k^s}{H(n,s)} \tag{1}$$

where $s > 0$ is the skewness parameter and

$$H(n,s) = \sum_{k=1}^{n} \frac{1}{k^s} \tag{2}$$

is the generalized harmonic number.

In this work, we calculate the average packet matching cost as [5]:

$$\min \sum_{i=1}^{n} f_i d_i \tag{3}$$

- f_i is calculated by the frequency of matched packets with that rule r_i.
- d_i is the depth of the rule in a given firewall policy.

3.2. Dependency Matrix

Not all permutations of R preserve the semantics of the firewall policy. Two rules r_i and r_j are considered dependent if exchanging their relative positions changes the classification result for at least one packet. Such dependencies usually appear when the matching conditions of the two rules overlap, meaning that at least one packet may satisfy both rules. To address these ordering constraints explicitly, we define a binary dependency matrix D of size $n \times n$ such that:

$$D[i][j] = \begin{cases} 1 & \text{if } r_i \text{ must appear before } r_j \text{ in any valid ordering} \\ 0 & \text{otherwise} \end{cases}$$

The matrix D defines a partial order over the rule set. Any valid firewall ordering must therefore respect the dependency matrix. As a result, the set of feasible orderings corresponds to the set of all linear extensions of this partial order.

https://doi.org/10.3390/engproc2026168020

Definition 1 (Dependency density)**.** *If we consider d as the number of dependency constraints in a policy of n rules [17]. The dependency density ρ is defined as:*

$$\rho = \frac{d}{n(n-1)/2} \tag{4}$$

where $n(n-1)/2$ is the total number of possible pairs (i, j) such that $i < j$. The value of ρ varies from 0, with no ordering constraints, to 1, when all pairs are constrained and only one valid ordering remains.

3.3. Freedom Rate

In order to measure the proportion of rule pairs that are free to be reordered, we define the freedom rate as follow, for a dependency matrix D of size $n \times n$ [11].

$$F = \frac{|\{(i,j) \mid i \neq j,\ D[i][j] = 0,\ D[j][i] = 0\}|}{n(n-1)} \tag{5}$$

The numerator counts all ordered pairs (i, j) for which no dependency exists in either direction, while the denominator gives the total number of ordered pairs. The freedom rate varies from 0, when all pairs are constrained, to 1, when no dependency restricts the ordering. The freedom rate is directly related to the dependency density by:

$$F = 1 - 2\rho \tag{6}$$

The expression above shows that each dependency constraint removes two ordered pairs, namely (i, j) and (j, i), and reduces the available search space by a factor of $2/[n(n-1)]$.

Proposition 1. *The number of valid orderings means the number of linear extensions induced by D, decreases monotonically as F decreases. When $F = 1$ (equivalently, $\rho = 0$), all n! permutations are valid. When $F = 0$ (equivalently, $\rho = 1$), at most one valid ordering remains. The freedom rate therefore provides a simple and general measure of the effective search space available to any optimization method under the dependency constraints represented by D.*

3.4. Optimization Problem Statement

The firewall rule ordering problem with dependency constraint considered in this paper can now be formulated as follows: Given a set R of n rules, a dependency matrix D describing precedence constraints, and a Zipf traffic distribution with parameter s, determine an ordering π^* such that:

$$\pi^* = \arg \min_{\pi \in L(D)} C(\pi) = \arg \min_{\pi \in L(D)} \sum_{i=1}^{n} i \times f(\pi_i) \tag{7}$$

$$C(\pi) = \arg \min_{\pi \in L(D)} \sum_{i=1}^{n} i \times f(\pi_i) \tag{8}$$

where $L(D)$ denotes the set of all topological sorts of the directed acyclic graph induced by D, and $f(k)$ is the Zipf activation probability of rank k. This problem is NP-hard in general [2], since it includes the single-machine weighted scheduling problem with precedence constraints as a special case. In this work, we address it using a Discrete Cuckoo Search algorithm guided by the dependency matrix D, as described in the next section.

4. Reordering Discrete Cuckoo Search (RDCS)

In this section, we illustrate our optimization method (RDCS) using a running example. We consider a firewall policy composed of $n = 5$ rules, denoted by $R = \{r_0, r_1, r_2, r_3, r_4\}$. The dependency relations between rules are represented by the binary matrix D, where

$D[i][j] = 1$ means that rule r_i must appear before rule r_j in any valid ordering. In this example, the policy contains two precedence constraints: r_0 must precede r_2, and r_1 must precede r_3. All other elements of the matrix equal zero. Under a Zipf distribution with parameter $s = 1.2$, the frequency probabilities are approximately $f(r_0) = 0.105, f(r_1) = 0.184$, $f(r_2) = 0.441, f(r_3) = 0.069$, and $f(r_4) = 0.050$. Since r_2 has the highest probability value, an efficient ordering should place it as early as possible while respecting the dependency constraints. Therefore, any valid solution must satisfy the two conditions $r_0 \prec r_2$ and $r_1 \prec r_3$.

$$D = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

For the valid ordering $\pi = [r_0, r_2, r_1, r_4, r_3]$, the matching cost also known as the fitness value in this work, is calculated according to Table 1 as:

$$C(\pi) = 1 \times 0.441 + 2 \times 0.105 + 3 \times 0.184 + 4 \times 0.050 + 5 \times 0.069 = 1.748$$

Table 1. Example of a valid rule ordering for the running example.

Position	Rule	Activation Prob.	Cost Term	Constraint Status
1	r_0	0.441	1×0.441	$r_0 \prec r_2$ satisfied
2	r_2	0.105	2×0.105	after r_0
3	r_1	0.184	3×0.184	$r_1 \prec r_3$ satisfied
4	r_4	0.050	4×0.050	no constraint
5	r_3	0.069	5×0.069	after r_1

Discrete Cuckoo Search with Dependency Matrix

The Discrete Cuckoo Search (RDCS) algorithm maintains a population of valid orderings and improves them through three complementary phases inspired by cuckoo search behavior [10]. In our case, all generation and perturbation operations are guided by the dependency matrix D, which guarantees that every candidate solution remains a valid topological sort.

Population initialization: The initial population of size `POP_SIZE` is generated using constrained random topological sorting.

Perturbation operator: Given a solution π and a perturbation level k, the first k positions are preserved, while the remaining $k - n$ positions are rebuilt using Levy flight step [9]. This operator is used in all three phases of the algorithm, we basically implicitly divide the population into three groups best solutions, good solutions and bad solutions:

Phase A: Cuckoo phase (global exploration). Each best solution π^* is perturbed once to produce a new solution π'. If π' has a better fitness than a solution π in the population (typically among good solutions group), then π is replaced.

Phase B: Improvement phase (local exploitation). The good solutions are refined through `LOCAL_ITER` perturbations each. A perturbed solution is accepted only if it strictly improves the current one.

Phase C: Abandonment phase (diversification). The worst solutions in the population are deleted and replaced by new solutions that we obtained by applying the perturbation operator to the current best solution group. This produces additional diversity while keeping the search guided by the best known region.

https://doi.org/10.3390/engproc2026168020

Example: Consider the following simplified population of four valid solutions:

$$\begin{aligned}
\pi_1 &= [r_0, r_1, r_2, r_3, r_4], \quad C(\pi_1) = 1.656 \quad \text{(best)} \\
\pi_2 &= [r_0, r_1, r_4, r_2, r_3], \quad C(\pi_2) = 1.741 \\
\pi_3 &= [r_1, r_0, r_2, r_3, r_4], \quad C(\pi_3) = 1.773 \\
\pi_4 &= [r_1, r_0, r_4, r_2, r_3], \quad C(\pi_4) = 1.981 \quad \text{(worst)}
\end{aligned}$$

Phase A. Perturb π_1' with level $k = 1$, which fixes the prefix $[r_0]$. Suppose the reconstructed solution is:

$$\pi_1' = [r_0, r_4, r_1, r_2, r_3], \quad C(\pi_1') = 1.682$$

Since $1.682 < 1.773$, π_3 is replaced by π_1'.

Updated population:

$$\begin{aligned}
\pi_1 &= [r_0, r_1, r_2, r_3, r_4], \quad C(\pi_1) = 1.656 \\
\pi_2 &= [r_0, r_1, r_4, r_2, r_3], \quad C(\pi_2) = 1.741 \\
\pi_3 &= [r_0, r_4, r_1, r_2, r_3], \quad C(\pi_3) = 1.682 \\
\pi_4 &= [r_1, r_0, r_4, r_2, r_3], \quad C(\pi_4) = 1.981
\end{aligned}$$

Phase B. Improve π_2 through several local perturbations. Assume one perturbation gives:

$$[r_0, r_1, r_2, r_4, r_3], \quad C(\pi_2) = 1.716$$

Since $1.716 < 1.741$, this new solution replaces π_2.

Phase C. Replace the worst solution π_4 by applying two successive perturbations to π_1. For example:

$$\pi_4 = [r_0, r_4, r_1, r_3, r_2]$$

followed by

$$\pi_4' = [r_0, r_1, r_3, r_4, r_2]$$

which becomes the new fourth individual.

These three phases combine global exploration, local improvement, and diversification. This balance makes RDCS suitable for constrained combinatorial optimization problems where the search space is restricted by dependency matrices of different densities.

5. Performance Evaluation

5.1. Experimental Setup

All experiments were conducted assuming that firewall policies were composed of $n = 100$ rules. For each run, the dependency matrix D was generated randomly using a distinct result in order to ensure statistical independence across repetitions. Each configuration was executed 30 times, and the reported values correspond to the averages over these runs. For the frequency of matching, we used Zipf's law with a skewness parameter $s = 1.2$, a value commonly used to represent non-uniform traffic distributions in network filtering scenarios. The RDCS algorithm was configured with a population size of 10, 500 iterations, a perturbation level of 2, and 10 local improvement attempts during each Phase B step. The competing methods; Random Search (RS) [14], Genetic Algorithm (GA) [13], and Simulated Annealing (SA) [18], were assigned the same computational budget, defined as `POP_SIZE` × `ITERATIONS` = 500 solution evaluations. For each repetition, all methods were tested on the same dependency matrix and with the same result in order to ensure a fair comparison. The dependency density was varied over eight levels:

$$\rho = \{0,\ 0.001,\ 0.002,\ 0.004,\ 0.008,\ 0.012,\ 0.016,\ 0.020\},$$

which correspond to 0 to 100 dependency constraints on 100 rules. The experimental study focuses on three aspects: the contribution of the dependency matrix, the comparison between optimization methods, and the influence of dependency density on solution quality.

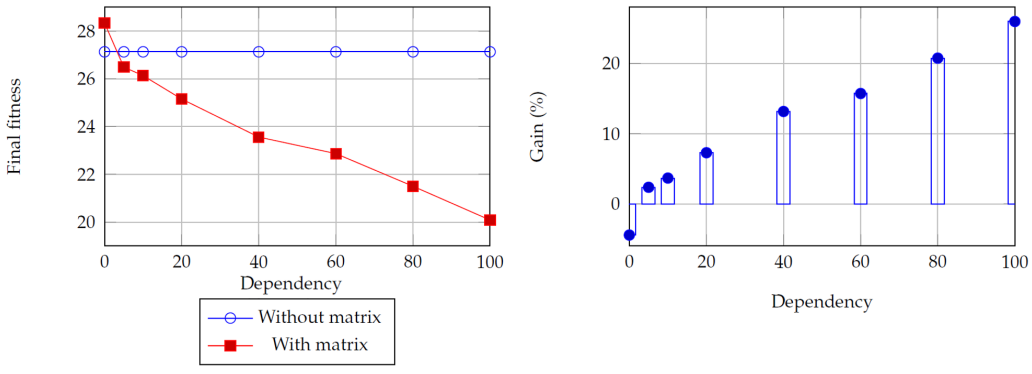
5.2. Impact of the Dependency Matrix

Table 2 compares RDCS without dependency constraints, where the search is carried out over free permutations, with RDCS guided by the dependency matrix D.

Table 2. Impact of the dependency matrix on RDCS performance.

Dependency	Density	RDCS Without Matrix		RDCS With Matrix		Gain (%)
		Fit.Fin	Impr.%	Fit.Fin	Impr.%	
0	0.0000	27.1300	23.55	28.3346	22.67	-4.44
5	0.0010	27.1300	23.55	26.4905	23.13	+2.36
10	0.0020	27.1300	23.55	26.1335	24.70	+3.67
20	0.0040	27.1300	23.55	25.1549	25.64	+7.28
40	0.0081	27.1300	23.55	23.5614	23.64	+13.15
60	0.0121	27.1300	23.55	22.8685	23.56	+15.71
80	0.0162	27.1300	23.55	21.5029	20.39	+20.74
100	0.0202	27.1300	23.55	20.0877	21.39	+25.96

When dependency density equals zero, both versions operate on the same unconstrained search space and therefore produce similar results. The small negative difference observed in this case is mainly due to the perturbation mechanism rather than to the impact of constraints. Starting From `dep` $= 5$, the matrix-guided version consistently achieves better final fitness, and the gain increases gradually with the number of dependency constraints, from +2.36% to +25.96%. These results show that the dependency matrix does not simply reduce the search space; it also improves the search process by directing the algorithm toward feasible and relevant orderings from the beginning. On the other hand, the unconstrained version spends part of its budget exploring permutations that would not satisfy the policy constraints in practice. The increasing gain observed at higher densities as shown in Figure 1 indicates that the value of the dependency matrix becomes stronger as the policy structure becomes more constrained. we can also mention that the final fitness of RDCS without the matrix remains constant at 27.13 for all density levels. This behavior is consistent with the experimental design, since this version does not use D and is therefore insensitive to the number of dependencies.



(**a**) Final fitness (**b**) Gain of the dependency matrix

Figure 1. Effect of the dependency matrix on RDCS performance.

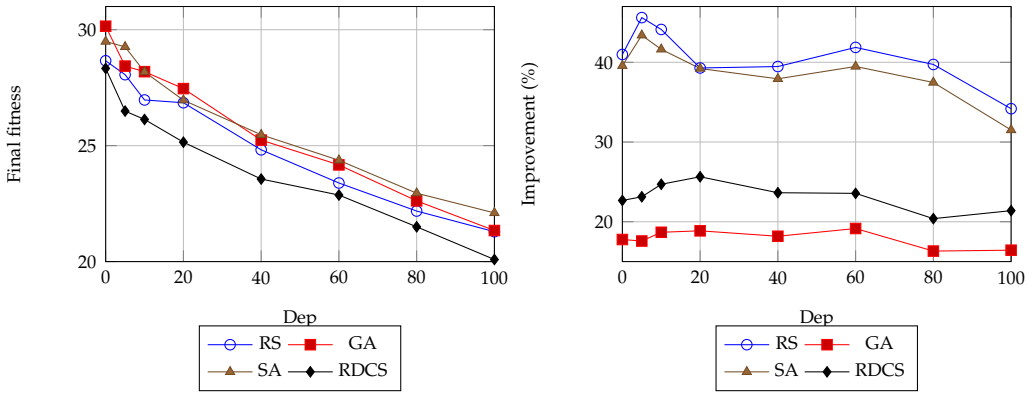
5.3. Comparison with Competing Methods

Table 3 compares RS, GA, SA, and RDCS under identical conditions, namely the same dependency matrix, the same result, and the same computational budget.

Among all tested density levels shown in Figure 2 RDCS gives the lowest final fitness, which indicates better overall performance than RS, GA, and SA under the same conditions. Compared with GA, the gain of RDCS ranges from +4.94% to +8.39%, with the largest gap observed at `dep` = 20. This can show that the RDCS search strategy is better adapted to constrained permutation problems than the crossover and mutation operators used in GA. GA shows the lowest improvement percentages, between 16% and 19%, which suggests limited efficiency within the available budget. SA reaches higher improvement rates, between 31% and 44%, but still ends with higher final fitness values than RDCS. This indicates that SA can escape local optima, but does not exploit the most promising valid regions of the search space as effectively as RDCS. This behavior can be related to the three-phase design of RDCS. The first phase promotes global exploration around the best valid solution, the second phase performs local refinement, and the third phase restores diversity while remaining inside the feasible search space. This balance appears particularly suitable for firewall rule ordering under dependency constraints.

Table 3. Comparison of methods under identical conditions.

Dep	Dens.	RS		GA		SA		RDCS	
		Fit	Impr	Fit	Impr	Fit	Impr	Fit	Impr
0	0.0000	28.66	40.98	30.15	17.77	29.49	39.57	28.33	22.67
5	0.0010	28.06	45.61	28.43	17.58	29.26	43.39	26.49	23.13
10	0.0020	26.97	44.12	28.19	18.68	28.18	41.63	26.13	24.70
20	0.0040	26.85	39.29	27.46	18.86	26.96	39.22	25.15	25.64
40	0.0081	24.82	39.47	25.24	18.18	25.47	37.93	23.56	23.64
60	0.0121	23.39	41.87	24.17	19.15	24.37	39.48	22.87	23.56
80	0.0162	22.18	39.73	22.62	16.32	22.95	37.47	21.50	20.39
100	0.0202	21.29	34.18	21.34	16.43	22.10	31.51	20.09	21.39



(**a**) Final fitness (**b**) Improvement rate

Figure 2. Comparison of RS, GA, SA, and RDCS under identical conditions.

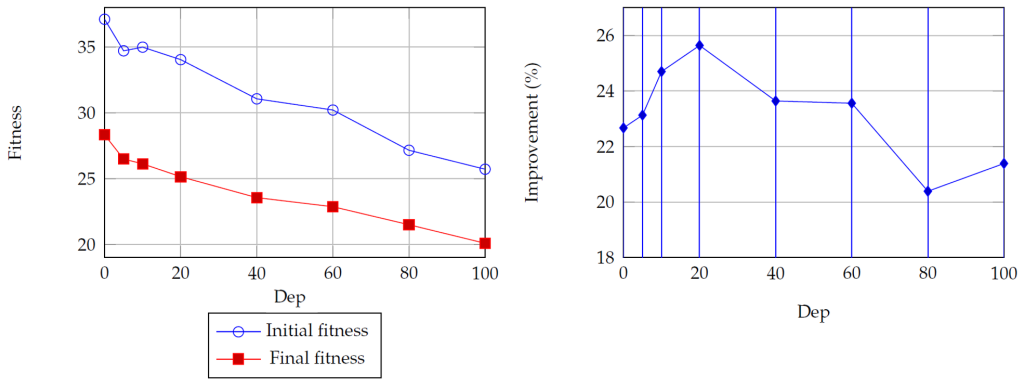
5.4. Effect of Dependency Density on Solution Quality

Table 4 reports the evolution of RDCS performance as dependency density increases, together with the standard deviation of the improvement over 30 repetitions. Two main observations can be drawn from Table 4. First, both the initial and final fitness values decrease as dependency density increases. The initial fitness drops from 37.12 at `dep` = 0

to 25.72 at dep = 100, while the final fitness decreases from 28.33 to 20.09. Overall, this corresponds to a 29.1% reduction in the cost of the packet matching. This result shown in Figure 3 reflects the constructive effect of the dependency matrix: by imposing precedence constraints, it can place some highly activated rules earlier and thus improve the baseline quality of feasible solutions. Second, the improvement percentage remains relatively stable, fluctuating between about 20% and 26%, without a clear monotonic pattern. This indicates that dependency density strongly affects the absolute quality of the obtained solutions, but has a limited effect on the relative ability of RDCS to improve its starting point. In other words, the algorithm keeps a stable optimization capacity over all tested density levels. The standard deviation, ranging from 7.4 to 9.6, mainly reflects the diversity of the randomly generated dependency structures rather than the instability of the algorithm itself. Since each repetition uses a different matrix, the effective search space varies from one run to another. Even when the observed gain is moderate in some cases, its effect remains important in practice. In real-time packet filtering, small reductions in matching cost can lead to significant cumulative benefits in throughput and response time, especially in large-scale and security-sensitive network environments. For this reason, dependency-aware rule ordering should be considered an essential part of firewall optimization.

Table 4. Impact of dependency density on RDCS quality.

Dep	Density	Freedom	Fit. Init	Fit. Final	Impr. (%) ± std
0	0.0000	1.0000	37.12	28.33	22.67 ± 9.56
5	0.0010	0.9990	34.72	26.49	23.13 ± 6.94
10	0.0020	0.9980	35.00	26.13	24.70 ± 8.30
20	0.0040	0.9960	34.05	25.15	25.64 ± 8.66
40	0.0081	0.9919	31.07	23.56	23.64 ± 7.61
60	0.0121	0.9879	30.22	22.87	23.56 ± 8.61
80	0.0162	0.9838	27.16	21.50	20.39 ± 7.41
100	0.0202	0.9798	25.72	20.09	21.39 ± 7.44



(**a**) Initial and final fitness (**b**) Improvement rate with standard deviation

Figure 3. Effect of dependency density on RDCS quality.

6. Conclusions

This study explored the firewall rule ordering problem under dependency constraints and found that the dependency matrix plays a crucial role in the optimization process. The experimental results indicate that using the matrix improves the quality of the solutions by guiding the search toward valid rule orderings and avoiding infeasible permutations. This effect becomes more visible as dependency density increases, where the gain of the matrix guided version over the unconstrained version grows steadily. The comparison with the

other methods also shows that RDCS achieves the best final fitness under all tested configurations. In addition, while dependency density changes the absolute quality of the solutions, the optimization capability of RDCS remains stable across the different levels of constraints. These results confirm that the dependency matrix is not only a formal representation of rule precedence, but also a practical component of the optimization method.

Author Contributions: Conceptualization, F.B. and D.L.; methodology, F.B. and D.L.; software, F.B.; validation, F.B. and D.L.; formal analysis, F.B.; investigation, F.B.; data curation, F.B.; writing—original draft preparation, F.B.; writing—review and editing, F.B. and D.L.; visualization, F.B.; supervision, D.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The synthetic data generated and analyzed during this study are available from the corresponding author upon reasonable request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Alsmadi, I. Misconfiguration in Firewalls and Network Access Controls: Literature Review. *Future Internet* **2021**, *13*, 283. [CrossRef]
2. Fulp, E.W. Optimization of Network Firewall Policies using Directed Acyclical Graphs. In *Proceedings of the IEEE Internet Management Conference*; Citeseer: Princeton, NJ, USA, 2005.
3. Dobrescu, M.; Argyraki, K. Dynamic Rule-Ordering Optimization for High-Speed Firewall Filtering. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security*; Association for Computing Machinery: New York, NY, USA, 2006.
4. Horn, W.A. Single-Machine Job Sequencing with Treelike Precedence Ordering and Linear Delay Penalties. *SIAM J. Appl. Math.* **1972**, *23*, 189–202. [CrossRef]
5. Al-Shaer, E.S.; Hamed, H.H. Discovery of Policy Anomalies in Distributed Firewalls. In *Proceedings of the IEEE INFOCOM 2004*; IEEE: Piscataway, NJ, USA, 2004; Volume 23, pp. 2605–2616.
6. Abedin, M.A.U.; Nessa, S.; Khan, L.; Thuraisingham, B. Detection and Resolution of Anomalies in Firewall Policy Rules. In *Proceedings of the Data and Applications Security XX*; Springer: Berlin/Heidelberg, Germany, 2006; LNCS; Volume 4127. [CrossRef]
7. Eppstein, D.; Muthukrishnan, S. Internet Packet Filter Management and Rectangle Geometry. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*; Association for Computing Machinery: New York, NY, USA, 2001; pp. 827–835.
8. El-Alfy, E.S.M. A Heuristic Approach for Firewall Policy Optimization. In *Proceedings of the ICACT 2007*; IEEE: Piscataway, NJ, USA, 2007; pp. 851–856.
9. Bezzazi, F.; Lotfi, D. Firewall Rule Ordering Optimization Using Discrete Cuckoo Search. In *Proceedings of the WINCOM*; IEEE: Piscataway, NJ, USA, 2025; pp. 1–6.
10. Yang, X.S.; Deb, S. Cuckoo Search via Lévy Flights. In *Proceedings of the World Congress on Nature and Biologically Inspired Computing (NaBIC 2009)*; IEEE: Piscataway, NJ, USA, 2009; pp. 210–214.
11. Ben Neji, N.; Bouhoula, A. Towards Safe and Optimal Filtering Rule Reordering for Complex Packet Filters. In *Proceedings of the 2011 5th International Conference on Network and System Security (NSS)*; IEEE: Piscataway, NJ, USA, 2011; pp. 153–160. [CrossRef]
12. Fonseca, N.; Moura, P.; Carvalho, P. An innovative two-stage algorithm to optimize firewall rule ordering. *Comput. Secur.* **2023**, *134*, 103434. [CrossRef]
13. Holland, J.H. *Adaptation in Natural and Artificial Systems*, 2nd ed.; MIT Press: Cambridge, MA, USA, 1992.
14. Kirkpatrick, S.; Gelatt, C.D.; Vecchi, M.P. Optimization by Simulated Annealing. *Science* **1983**, *220*, 671–680. [CrossRef] [PubMed]
15. Ouaarab, A. *Discrete Cuckoo Search for Combinatorial Optimization*; Springer: Berlin/Heidelberg, Germany, 2020. [CrossRef]
16. Zipf, G.K. *Human Behavior and the Principle of Least Effort*; Addison-Wesley: Cambridge, MA, USA, 1949.

17. Bang-Jensen, J.; Gutin, G. *Digraphs: Theory, Algorithms and Applications*, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 2008.
18. Zhigljavsky, A. Random Search for Global Optimization. In *Encyclopedia of Optimization*; Pardalos, P.M., Prokopyev, O.A., Eds.; Springer: Cham, Switzerland, 2023. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.