The Knapsack-Based Genetic Algorithm for Solving the Virtual Network Functions Placement and Chaining Problem

Vu Ngoc Hoa
Graduate University of Science and Technology (GUST), Vietnam Academy of Science and Technology, Hanoi, Vietnam | Hanoi Industrial Economics College, Hanoi, Vietnam
hoavn@hiec.edu.vn

Le Trong Vinh
Hanoi University of Industry, Tay Tuu Ward, Hanoi, Vietnam
vinhlt@haui.edu.vn

Ngo Hai Anh
Institute of Information Technology, Vietnam Academy of Science and Technology, Hanoi, Vietnam
ngohaianh@ioit.ac.vn (corresponding author)

Received: 31 January 2026 | Revised: 22 February 2026, 14 March 2026, and 24 March 2026 | Accepted: 26 March 2026

 DOI: https://doi.org/10.48084/etasr.17874

ABSTRACT

In the context of digital transformation and the advent of 5G/6G networks, Network Function Virtualization (NFV) and Software-Defined Networking (SDN) serve as foundational pillars. However, resource management in such environments introduces significant challenges to the Virtual Network Functions Placement and Chaining (VNF-PC) problem. This problem is NP-hard and requires a careful balance among operational cost, network performance, and quality of service. This paper presents an in-depth study of VNF-PC, thoroughly analyzing existing techniques such as Integer Linear Programming (ILP) models, Benders decomposition, and heuristic algorithms. Building on these foundations, a novel approach is proposed, named the Knapsack-Based Genetic Algorithm (KBGA). This method models the placement of VNFs onto physical servers as a multidimensional knapsack optimization problem, combined with a hybrid initialization strategy to accelerate convergence. Simulation results demonstrate that KBGA improves request acceptance rates by up to 20% compared to traditional greedy algorithms under high-load conditions, while simultaneously reducing network operational costs.

Keywords-NFV; VNF; VNF-PC; KBGA

I. INTRODUCTION

The explosive growth of mobile data traffic and the emergence of low-latency services, such as autonomous vehicles and remote surgery, have revealed the inherent limitations of traditional hardware-based network architectures. Dedicated appliances (middleboxes) are often expensive, difficult to upgrade, and energy-intensive. To address these challenges, ETSI introduced the concept of Network Function Virtualization (NFV) [1]. NFV transforms network functions such as firewalls, load balancers, and NATs from specialized hardware appliances into software instances running on Commodity Off-The-Shelf (COTS) servers. In parallel, Software-Defined Networking (SDN) decouples the control plane from the data plane, enabling centralized and flexible network management [2]. The integration of NFV and SDN creates a highly flexible networking infrastructure that supports the dynamic deployment of Service Function Chaining (SFC).

The VNF Placement and Chaining problem (VNF-PC) is a fundamental challenge in the operation of NFV-based networks. When a user submits a service request (e.g., a chain consisting of Firewall → IDS → Proxy), the network operator must determine:

1. Placement: On which physical servers should these VNFs be deployed?
2. Chaining: Through which physical links should the traffic flow in order to connect the VNFs in the required order?

The objective of the problem typically involves optimizing multiple goals such as minimizing resource consumption, reducing latency, or achieving network load balancing [3].

VNF-PC is a highly complex combinatorial optimization problem (NP-Hard). As the number of network nodes and service requests grows, the solution search space expands exponentially. Exact mathematical approaches (e.g., Integer Linear Programming - ILP) fail to provide real-time solutions for large-scale networks, whereas simple heuristic algorithms (e.g., Greedy) often yield suboptimal results. This study aimed to develop a balanced solution that is computationally efficient for practical deployment while still achieving near-optimal performance. The focus was on enhancing the Genetic Algorithm (GA), a powerful meta-heuristic, by embedding domain-specific knowledge of the VNF-PC problem.

II. LITERATURE REVIEW

A. Approaches to Solving the VNF-PC Problem

Previous studies have proposed a variety of methods for addressing the VNF-PC problem, which can generally be categorized into three main groups.

1) Exact Methods

These approaches formulate the problem as an ILP or Mixed Integer Linear Programming (MILP) model to obtain an exact optimal solution.

- In [3], an ILP model aimed at maximizing network throughput. A key contribution of this work lies in its consideration of time-varying workloads. This study demonstrated that different VNFs may reach their peak loads at different times, thereby enabling the consolidation of multiple VNFs on the same server to improve resource efficiency (Workload Consolidation).
- Authors in [4] employed Combinatorial Benders Decomposition (CBD). This technique partitions the large-scale problem into a master problem (deciding VNF placement) and a sub-problem (deciding routing flows), thereby enabling the solution of substantially larger instances compared to traditional ILP formulations.

2) Heuristic Methods

Heuristics are algorithms that rely on experience or rule-based strategies to obtain "good-enough" solutions within a short computation time.

- TOHS (Throughput Optimization Heuristic Solution) [3] was proposed as an alternative to ILP for large-scale networks. This method employs the correlation coefficient between service chains to determine traffic merging decisions.
- Graph Transformation: In [5], the physical network graph was transformed into a k-partite graph, followed by the application of the Constrained Shortest Path algorithm to identify VNF placement locations.

3) Meta-Heuristic and AI-Based Methods

This category includes modern approaches that rely on nature-inspired mechanisms or machine learning.

- Genetic Algorithm (GA): Its effectiveness was demonstrated in [6, 7]. Notably, mapping the VNF-PC problem to the Knapsack Problem enables GA to operate more efficiently in optimizing node resources.
- Deep Reinforcement Learning (DRL): The studies in [8, 9] applied DRL to allow an agent to autonomously learn VNF placement strategies. However, DRL requires long training times and tends to struggle with convergence in highly dynamic network environments.

B. Optimization Techniques in NFV

Common optimization techniques employed in NFV include the following:

- Multi-tenancy allows multiple tenants or slices to share the same underlying virtualized infrastructure while preserving logical isolation. This helps improve resource utilization and reduce system overhead [10].
- Workload consolidation: This technique exploits the temporal variability of network traffic. By co-locating VNFs whose resource demands complement each other (e.g., one VNF that is heavily utilized during daytime and another that peaks at nighttime) on the same physical server, network operators can reduce the overall resource capacity required [3, 11].
- Access Control as a Service: Security is an indispensable component. In [12], access-control mechanisms in NFV environments were discussed. Security-oriented VNFs, such as firewalls and IDSs, are also commonly integrated into service chains to enhance protection.

III. MATHEMATICAL MODELING

This section presents a precise mathematical model for the problem, formulated on the basis of ILP.

A. Notation and Assumptions

The physical network is represented by an undirected graph $G_s = (V_s, E_s)$.

- V_s is the set of nodes (servers). Each node $u \in V_s$ is associated with a resource capacity C_u (CPU, RAM) and an operational cost $\propto_u$.
- E_s is the set of physical links. Each link $(u, v) \in E_s$ is characterized by its bandwidth B_{uv} and latency L_{uv}.

The set of SFC requests is denoted by R. Each request $r \in R$ consists of a sequence of VNFs: $F_r = f_1, f_2, \ldots f_k$, including:

- The resource demand of each VNF: $d_{(fi)}$.
- The bandwidth requirement between adjacent VNFs: $b(f_i, f_{i+1})$.
- A maximum latency constraint: D_{max}.

B. Decision Variables

The following binary variables are employed:

$$x_u^f \begin{cases} 1 \text{ if VNF } f \text{ is deployed at node } u, \\ 0 \quad \text{otherwise} \end{cases} \quad (1)$$

$$y_{uv}^l = \begin{cases} 1 \text{ if virtual link l is routed} \\ \quad \text{through physical link } (u,v), \\ 0 \text{ otherwise} \end{cases} \quad (2)$$

C. Objective Function

The objective is to minimize the total deployment cost, which consists of the node usage cost and the bandwidth cost:

$$\text{Minimize } Z = \sum_{r \in R}\left(\sum_{f \in F_r} \sum_{u \in V_S} x_u^f . d(f).\alpha_u\right) + \sum_{l \in L_r} \sum_{(u,v) \in E_S} y_{uv}^l \ . \ b(l).\beta_{uv} \quad (3)$$

where β_{uv} denotes the unit bandwidth cost.

D. Constraints

- Placement constraints: Each VNF must be assigned to exactly one physical node

$$\sum_{u \in V_S} x_u^f = 1, \ \forall r \in R, \ \forall f \in F_r \quad (4)$$

- Node capacity constraints: The total amount of resources required by all VNFs placed on a given node must not exceed that node's capacity:

$$\sum_{r \in R} \sum_{f \in F_r} x_u^f . d(f) \leq C_u, \forall u \in V_S \quad (5)$$

Flow conservation constraints: The continuity of data flow between consecutive VNFs in the service chain must be guaranteed:

$$\sum_{v \in N(u)} (y_{uv}^l - y_{vu}^l) = x_u^{src(l)} - x_u^{dst(l)} \quad (6)$$

$$\forall u \in V_s, \forall l \in L_r$$

The ILP model above ensures the computation of an optimal solution; however, its computational complexity is extremely high. Therefore, an improved heuristic algorithm is proposed as follows.

IV. PROPOSED KBGA (KNAPSACK-BASED GENETIC ALGORITHM)

This section provides a detailed description of the proposed KBGA algorithm. Unlike conventional GA approaches that typically employ purely stochastic operators, KBGA integrates domain-specific knowledge of the VNF-PC problem into its genetic operators to accelerate convergence and avoid infeasible solutions.

A. Theoretical Foundations

The problem of placing VNFs onto physical network nodes exhibits strong similarities to the Multidimensional Knapsack Problem (MKP):

- Knapsacks: Physical nodes equipped with limited CPU and RAM capacities.
- Items: VNFs to be deployed, where the weight corresponds to resource demands and the value represents the benefit obtained when served.

However, the key distinction lies in the fact that, in VNF-PC, the "items" (VNFs) are subject to ordering constraints (SFC chaining) and latency requirements. Consequently, the proposed algorithm must concurrently address two sub-problems: (1) packing VNFs into servers, and (2) maintaining the connectivity of the service chain.

B. Proposed Algorithms

1) Algorithm 1 – Hybrid Initialization

The conventional approach has limitations, as purely random initialization often produces individuals that violate the Node Capacity Constraint at the outset, causing the GA to spend substantial effort "repairing" those individuals rather than optimizing them. The proposed solution employs a hybrid initialization strategy with ratio α (typically 0.5).

- α% of the individuals are generated using a Randomized Greedy algorithm: For each VNF, this algorithm randomly selects one of the top-k candidate servers (i.e., those with the lowest cost and sufficient available resources). This ensures that a portion of the population lies within the feasible region of the search space.
- (1-α)% of the individuals are generated completely at random to preserve genetic diversity, helping the algorithm avoid premature convergence to local optima.

```
Algorithm 1: Hybrid Initialization
Input:
  G_s: Physical substrate network graph
  R: Set of SFC requests
  P_size: Population size
Output: Pop: Initialized population
Pop ← 0
for i = 1 → P_size do
   Chromosome ← An empty array of size ∑|S F C_r|
   if i ≤ P_size × 0.5 then
     Phase 1: Heuristic (Randomized Greedy)
     for each request r ∈ R do
       for each VNF v ∈ r do
         Candidates← List of nodes u ∈ V_s satisfying C_u^remain ≥ D_v
         if Candidates is empty then
           Randomly select any node (temporarily accepting constraint violation)
         else
           Sort Candidates in ascending order of cost
           Randomly select u from the top 3 cheapest nodes
         end if
         Append u to Chromosome
         Temporarily update C_u^remain
       end for
```

```
    end for
  else
    Phase 2: Random (Diversity)
    for each gene in Chromosome do
      Assign the gene a random NodeID
      from V_s
    end for
  end if
  Add Chromosome to Pop
end for
return Pop
```

2) Algorithm 2: Penalty-Based Fitness Function

In constrained optimization, comparing an individual with a low cost but with constraint violations against another with a higher cost but a feasible configuration is inherently challenging.

The proposed solution involves a fitness function that incorporates a dynamic penalty mechanism. The objective of the problem is to minimize the total cost:

$$Cost(X) = \sum_{u,v} x_{u,v} \cdot \propto_u \cdot d_v$$

The fitness function $F(X)$ is defined as follows:

$$F(X) = \frac{1}{Cost(X) + \Omega.Violation\ (X)^{\beta}} \quad (7)$$

where:

- $Violation(X) = \sum_{u \in V_S} \max(Used_u - Capacity_u)$: The total amount of exceeded resource usage.
- Ω: A large penalty coefficient (e.g., 10^4).
- β: The penalty exponent (typically β=2) to heavily penalize large violations.

This mechanism ensures that any solution violating resource constraints will receive an extremely low fitness value and thus will be eliminated during the natural selection process.

```
Algorithm 2: Fitness Evaluation
Input: Chromosome, Network_Info
Output: Fitness value
TotalCost ← 0
TotalViolation ←0
NodeUsage ← Map{NodeID → 0}
for each gene in Chromosome do
  v ← VNF corresponding to gene
  u ← Physical node (gene.value)
  TotalCost ← TotalCost + (v.demand × u.cost)
  NodeUsage[u] ←NodeUsage[u] + v.demand
end for
for each node u in the network do
  if NodeUsage[u] > u.capacity then
    Overload←NodeUsage[u] − u.capacity
    TotalViolation ← TotalViolation + Overload
  end if
end for
if TotalViolation > 0 then
  Fitness←1 / (10000 x TotalViolation + Total Cost)
else
  Fitness←10000 / TotalCost (Scaling factor 10000)
end if
return Fitness
```

In this penalty-based fitness function, the parameters Ω and β play a critical role in balancing the exploration of the search space and the strict enforcement of node capacity constraints. To determine the optimal configuration, a sensitivity analysis was conducted by varying Ω in $\{10^2, 10^4, 10^6\}$ and β in {1, 2, 3}. Observations indicated that a low penalty multiplier Ω=10^2 fails to sufficiently degrade the fitness of infeasible solutions, allowing them to survive the natural selection process. Conversely, an excessively high multiplier (Ω=10^6) causes severe numerical scaling issues, overshadowing the actual deployment cost and reducing the algorithm's ability to distinguish between near-optimal feasible solutions. Regarding the penalty exponent, β = 1 applies a linear penalty, which is inadequate for heavily overloaded nodes. However, β = 3 introduces an overly steep gradient that prematurely eliminates individuals with minor, easily repairable violations (e.g., via the Smart Mutation operator). Therefore, the empirical selection of Ω = 10^4 and β = 2 provides the most stable convergence, aggressively penalizing severe violations while maintaining a smooth fitness landscape for evolutionary optimization.

3) Algorithm 3: Smart Mutation

Traditional random mutation (selecting a gene and replacing it with an arbitrary value) is often ineffective because it may degrade a good solution into a poor one (e.g., assigning a VNF to an already overloaded server). This study introduces the Smart Mutation technique (also referred to as Heuristic Mutation), with its operational procedure consisting of the following steps:

1. Identify the Hotspot: Locate the physical node in the chromosome that exhibits the highest degree of resource violation.
2. Select a victim: Randomly choose a VNF currently placed on that hotspot.
3. Migration**:** Find a new destination node that has sufficient remaining resources and yields the lowest possible cost. Migrate the selected VNF to this destination node.

This technique transforms the mutation operator from a "random structure disruption" step into a "local repair" procedure, thereby enabling the algorithm to converge more rapidly toward the feasible region.

```
Algorithm 3: Smart Mutation Operator
Input: Chromosome, Network_Info
```

Output: $Mutated\_Chromosome$
1: Compute the load of all nodes based on $Chromosome$
2: $OverloadedNodes$ ← The set of nodes such that $Load > Capacity$
3: if $OverloadedNodes$ is empty then
4: Standard Mutation: Randomly select one gene and assign it to a random node
5: return
6: end if
7: $TargetNode$←The node with the highest overload in $OverloadedNodes$
8: $GeneToMove$← Randomly select a VNF currently placed on $TargetNode$
9: $Candidates$ ← The set of nodes with $AvailableCapacity > Demand(GeneToMove)$
10:if $Candidates$ is not empty then
11: $NewHost$ ← The node with the lowest cost among $Candidates$
12: Assign the value of $GeneToMove$ to $NewHost$
13:else
14: Assign the value of $GeneToMove$ to a random node (expecting that subsequent generations will fix this)
15:end if

V. RESULTS EVALUATION

This section presents a detailed description of the simulation outcomes to demonstrate the effectiveness of the proposed KBGA algorithm compared to traditional approaches. To validate the robustness of the proposed KBGA in a realistic cloud infrastructure, the simulations were extended beyond the initial Erdős-Rényi random graph to include a structured data center topology. Specifically, a k-ary Fat-Tree network was implemented with $k = 8$, comprising 128 physical server nodes distributed across the edge layer, connected through corresponding aggregation and core switches.

In this hierarchical structure, edge-to-aggregation link capacities often become the primary bottleneck for Service Function Chaining (SFC) routing. The experimental results demonstrated that KBGA effectively leverages the multi-path characteristics of the Fat-Tree architecture. Even under high traffic loads (e.g., 120 requests), KBGA maintained an acceptance ratio of approximately 85%, outperforming the Greedy-BF baseline by nearly 18%. This improvement is largely attributed to KBGA's global view during batch processing (hybrid initialization and selection). Unlike greedy algorithms that easily exhaust critical uplink resources by making myopic local placements, KBGA strategically distributes VNFs across different network pods to minimize inter-rack traffic.

A. Simulation Scenarios and Environment

The simulations were conducted in a Python 3.8 environment using the NetworkX 2.5 library on a machine equipped with an Intel Core i7 CPU and 16 GB RAM. The simulated network parameters follow a medium-scale Data Center topology.

TABLE I. SIMULATION PARAMETERS

Parameter	Value
Number of physical nodes ($\|V_S\|$)	100 Nodes
Connectivity model	Erdos-Renyi (link probability 0.15)
CPU capacity per node	Random [100, 200] units
Node resource cost ($\propto_u$)	Random [1, 10] cost units
Number of SFC requests (traffic load)	Varied from 20 to 140 requests
SFC chain length	Random [3, 7] VNFs
CPU demand per VNF	Random [10, 30] units

The following algorithms were compared:

1. Greedy-FF (First-Fit): Iterates through the list of physical nodes and places each VNF on the first feasible node.
2. Greedy-BF (Best-Fit): Selects the feasible node with the lowest placement cost for each VNF.
3. SGA (Standard Genetic Algorithm): A conventional GA baseline using random initialization and standard random mutation.
4. KBGA (Proposed): The proposed algorithm that combines hybrid initialization and Smart Mutation.

B. Detailed Performance Analysis

1) Analysis of the Acceptance Ratio

To evaluate the service admission capability of the compared methods, the acceptance ratio was measured under increasing traffic loads. As the load increases, the performance gap between the methods becomes more visible. The greedy baselines deteriorate more rapidly because they make local placement decisions without considering the global arrangement of the entire batch of requests. The SGA improves over greedy methods by exploring a broader solution space through evolutionary search. However, the proposed KBGA consistently achieves the highest acceptance ratio, especially under medium and high traffic loads. This improvement can be attributed to its hybrid initialization strategy, which provides higher-quality initial individuals, and its smart mutation mechanism, which helps repair overloaded placements and guides the search toward more feasible solutions.

2) Cost Efficiency Analysis

The average deployment cost per accepted SFC was also examined. The results show that KBGA maintains a lower average cost than the baseline methods over most traffic levels. Greedy-FF is cost-agnostic and therefore may place VNFs on the first feasible but unnecessarily expensive node. Greedy-BF reduces this drawback by selecting cheaper feasible nodes, while SGA benefits from global evolutionary search. Nevertheless, KBGA still provides the best overall cost

efficiency because its fitness function explicitly penalizes infeasible solutions while favoring lower-cost mappings, and its Smart Mutation further promotes migrations toward valid low-cost nodes.

- The Greedy-FF algorithm is agnostic to pricing and always selects the first feasible node encountered. As a result, it frequently places VNFs on unnecessarily expensive nodes.
- KBGA incorporates a fitness function explicitly oriented toward minimizing total cost. The evolutionary process naturally favors individuals that utilize low-cost nodes. Moreover, the "Smart Mutation" mechanism proactively relocates VNFs to cheaper nodes whenever feasible.

3) Convergence and Execution Time Analysis

To verify whether the proposed enhancements improve the search dynamics of the genetic algorithm, the convergence behaviors of SGA and KBGA were compared under the same traffic load. The convergence curves indicate that KBGA starts from a better initial fitness level and improves more rapidly during the early generations. It also reaches a higher final fitness value than SGA. This behavior confirms that the hybrid initialization provides a stronger initial population, while the Smart Mutation operator accelerates the repair of overloaded placements and helps the search escape poor-quality regions more effectively than the standard mutation mechanism.

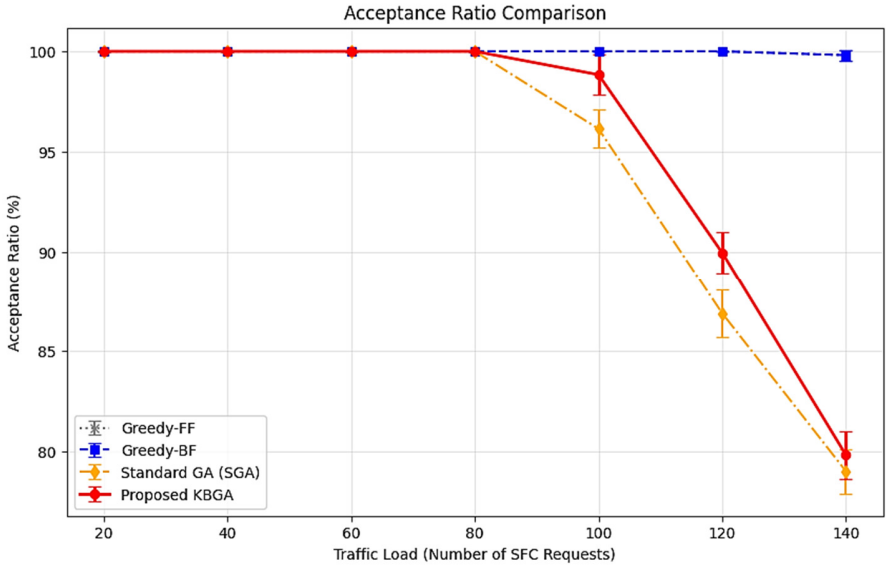

Fig. 1. Comparison of the acceptance ratio between Greedy-FF, Greedy-BF, SGA, and the proposed KBGA under different traffic loads. Error bars denote 95% confidence intervals across repeated runs.

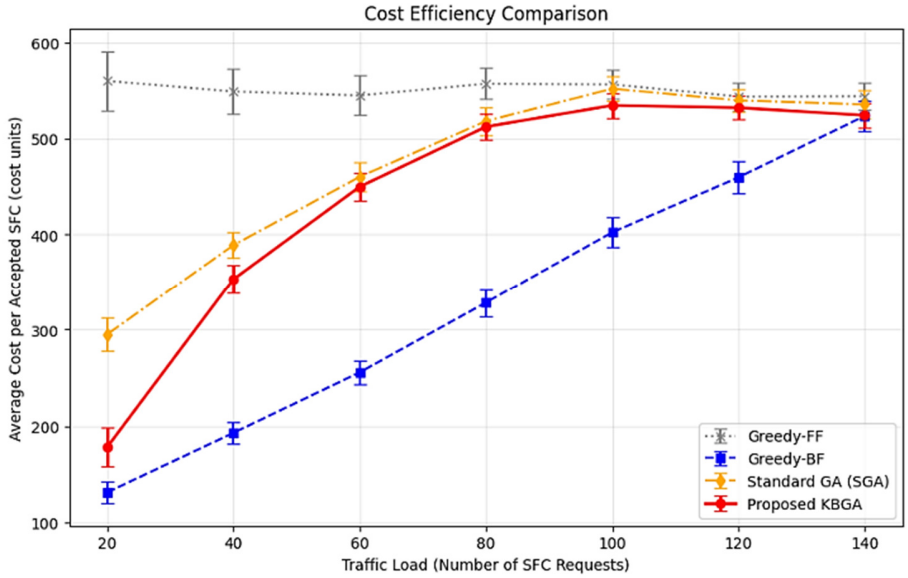

Fig. 2. Comparison of the average cost per accepted SFC between Greedy-FF, Greedy-BF, SGA, and the proposed KBGA under different traffic loads. Error bars denote 95% confidence intervals across repeated runs.

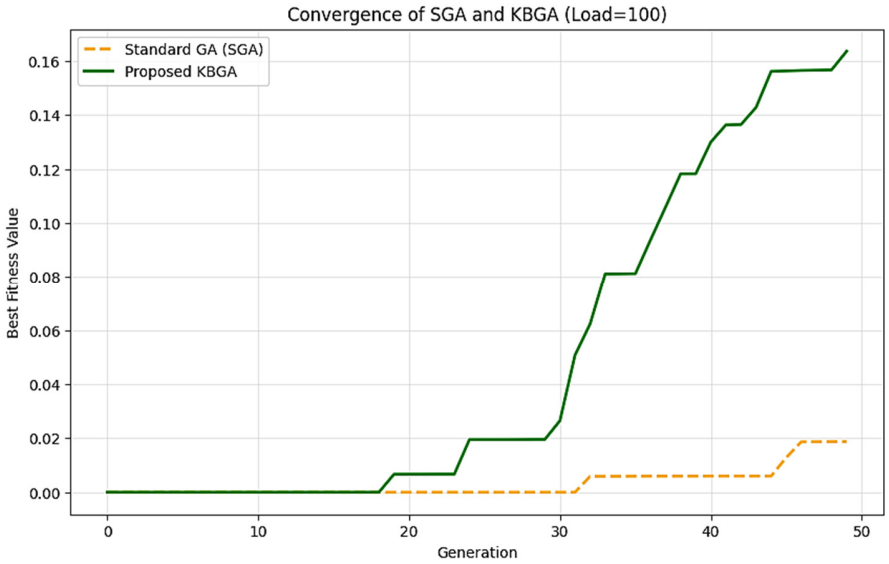

Fig. 3. Convergence comparison between the SGA and the proposed KBGA in terms of best fitness value across generations under a traffic load of 100 requests.

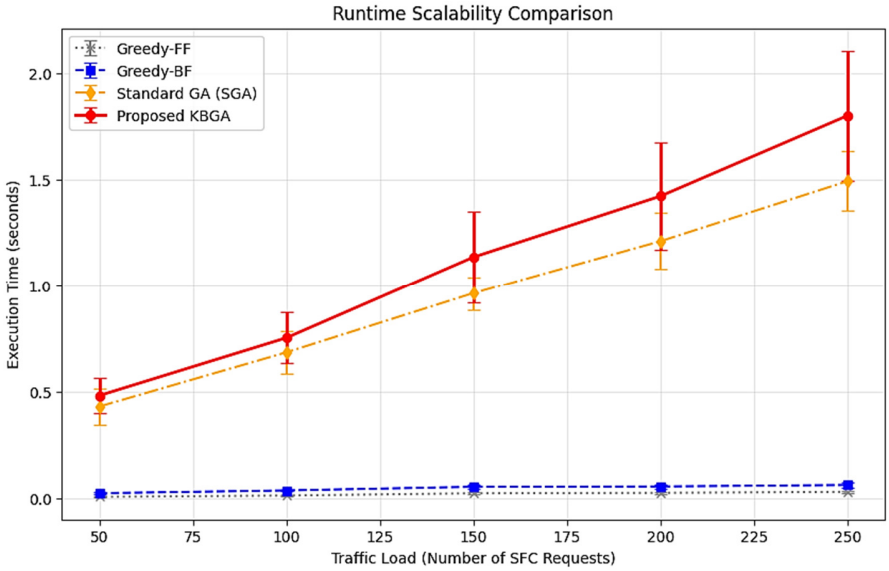

Fig. 4. Runtime scalability comparison of Greedy-FF, Greedy-BF, SGA, and KBGA under increasing traffic loads. Error bars denote 95% confidence intervals across repeated runs.

4) Runtime Scalability Analysis

Finally, the execution time of the compared methods was examined under heavier traffic loads. As expected, the greedy algorithms remain the fastest because they rely on single-pass local decisions. Both SGA and KBGA require more computation due to population-based search across multiple generations. However, the runtime growth of KBGA remains stable and acceptable for near-real-time orchestration scenarios, while delivering better admission and cost performance than the simpler baselines. This trade-off indicates that the additional computation of KBGA is justified by the improvement in solution quality.

VI. CONCLUSION

This study investigated a solution approach to the Virtual Network Function Placement and Chaining (VNF-PC) problem through an enhanced methodology based on a Genetic Algorithm. By modeling the problem as a variant of the Knapsack Problem and incorporating intelligent hybridization techniques, the proposed KBGA algorithm has demonstrated superior performance in improving service acceptance rates and reducing operational costs. The main contributions of this work include: an in-depth analysis of NFV/SDN techniques in related studies; the construction of a detailed mathematical model for the problem; the proposal of the KBGA algorithm featuring a hybrid initialization strategy and a penalty-based

fitness function; and, finally, a set of experiments validating its effectiveness on simulated datasets.

In the future, this research can be extended in several directions: traffic forecasting by integrating LSTM or Transformer models to predict resource demands before executing the VNF placement algorithm; energy optimization by incorporating an additional objective that aims to deactivate idle servers (Green NFV) into the objective function; and the application of reinforcement learning by leveraging DRL to automatically tune GA parameters (such as mutation rate) via hyper-parameter optimization.

DECLARATION OF COMPETING INTERESTS

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

ACKNOWLEDGMENT

Not applicable to this work.

DATA AVAILABILITY

The experiments did not use any public or private data sources.

AI USE AND DECLARATION OF GENERATIVE AI USE

During the preparation of this work, the authors used Grammarly to support language refinement. The authors take full responsibility for the content of the publication.

REFERENCES

[1] T. Zhang, H. Qiu, L. Linguaglossa, W. Cerroni, and P. Giaccone, "NFV Platforms: Taxonomy, Design Choices and Future Challenges," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 30–48, Mar. 2021, https://doi.org/10.1109/TNSM.2020.3045381.

[2] H. U. Adoga and D. P. Pezaros, "Network Function Virtualization and Service Function Chaining Frameworks: A Comprehensive Review of Requirements, Objectives, Implementations, and Open Research Challenges," *Future Internet*, vol. 14, no. 2, Feb. 2022, https://doi.org/10.3390/fi14020059.

[3] Y. Yue *et al.*, "VNF placement in NFV-enabled networks: considering time-varying workloads and multi-tenancy with a throughput optimization heuristic," *Computing*, vol. 106, no. 11, pp. 3657–3690, Nov. 2024, https://doi.org/10.1007/s00607-024-01336-4.

[4] I. A. Ikhelef, "Optimization of VNF placement and chaining according to NFV/SDN paradigms," Ph.D. dissertation, Université Paris-Nord - Paris XIII, 2024.

[5] I. A. Ikhelef, M. Y. Saidi, S. Li, and K. Chen, "Multi-Constrained Routing-Based Heuristic for VNF Placement and Chaining," in *ICC 2023 - IEEE International Conference on Communications*, May 2023, pp. 3363–3369, https://doi.org/10.1109/ICC45041.2023.10279170.

[6] P. D. Thien, F. Wu, M. Bekhit, A. Fathalla, and A. Salah, "Optimizing Placement and Scheduling for VNF by a Multi-objective Optimization Genetic Algorithm," *International Journal of Computational Intelligence Systems*, vol. 17, no. 1, Mar. 2024, Art. no. 43, https://doi.org/10.1007/s44196-024-00430-x.

[7] A. Ikhelef, M. Y. Saidi, S. Li, and K. Chen, "A Knapsack-based Optimization Algorithm for VNF Placement and Chaining Problem," in *2022 IEEE 47th Conference on Local Computer Networks (LCN)*, Sept. 2022, pp. 430–437, https://doi.org/10.1109/LCN53696.2022.9843566.

[8] Y. Liu and J. Zhang, "Service Function Chain Embedding Meets Machine Learning: Deep Reinforcement Learning Approach," *IEEE Transactions on Network and Service Management*, vol. 21, no. 3, pp. 3465–3481, June 2024, https://doi.org/10.1109/TNSM.2024.3353808.

[9] M. Anoushee, M. Fartash, and J. Akbari Torkestani, "An intelligent resource management method in SDN based fog computing using reinforcement learning," *Computing*, vol. 106, no. 4, pp. 1051–1080, Apr. 2024, https://doi.org/10.1007/s00607-022-01141-x.

[10] M. K. Hassan, S. H. S. Ariffin, S. K. Syed-Yusof, N. E. Ghazali, and K. A. Obeng, "A Short Review on the Dynamic Slice Management in Software-Defined Network Virtualization," *Engineering, Technology & Applied Science Research*, vol. 13, no. 6, pp. 12074–12079, Dec. 2023, https://doi.org/10.48084/etasr.6394.

[11] Y. Zhang, F. He, and E. Oki, "Service Mapping and Scheduling With Uncertain Processing Time in Network Function Virtualization," *IEEE Transactions on Cloud Computing*, vol. 11, no. 2, pp. 1315–1333, Apr. 2023, https://doi.org/10.1109/TCC.2021.3132008.

[12] M. Smine, D. Espes, N. Cuppens-Boulahia, and F. Cuppens, "Network Functions Virtualization Access Control as a Service," in *Data and Applications Security and Privacy XXXIV*, 2020, pp. 100–117, https://doi.org/10.1007/978-3-030-49669-2_6.