Optimizing Network Slices: A Comparative Analysis of Allocation Algorithms for 5G Environments

Christos Bouras
Computer Engineering and Informatics Department
University of Patras
Patras, Greece
Email: bouras@upatras.gr

Damianos Diasakos
Computer Engineering and Informatics Department
University of Patras
Patras, Greece
Email: up1084632@ac.upatras.gr

Apostolos Gkamas
Department of Chemistry
University of Ioannina
Ioannina, Greece
Email: gkamas@uoi.gr

Vasileios Kokkinos
Computer Engineering and Informatics Department
University of Patras
Patras, Greece
Email: kokkinos@cti.gr

Philippos Pouyioutas
Computer Science Department
University of Nicosia
Nicosia, Cyprus
Email: pouyioutas.p@unic.ac.cy

Nikolaos Prodromos
Computer Engineering and Informatics Department
University of Patras
Patras, Greece
Email: up1072549@ ac.upatras.gr

***Abstract*— In the realm of 5G networking, the optimization of user allocation through network slicing stands as a critical challenge, with the potential to substantially enhance the Quality of Service (QoS). This study examines three AI-based allocation algorithms—Simulated Annealing, which begins with a Randomized algorithm, Greedy, and Local Search with Hill Climbing—to efficiently distribute network resources. Next, we compare the algorithms for different user densities to understand how well each one can handle the situation at hand in terms of balance in allocation, consumption (time and memory) and complexity. Our research advances beyond conventional allocation techniques by offering different solutions for different needs thus improving QoS through the alignment of user demands with network capacity.**

Keywords*— Network Slicing, AI-Based Allocation Algorithms, 5G Quality of Service (QoS), Resource Optimization, Simulated Annealing

I. INTRODUCTION

The advent of 5G technology heralds a transformative era in telecommunications, distinguished by its capacity to deliver highly personalized network experiences through network slicing [1][2][3]. Network slicing allows for the segmentation of a single physical network into multiple virtual segments, each precisely tailored to meet specific user demands and service requirements. A critical challenge in this paradigm is the efficient allocation of users to these network slices, a factor that significantly impacts network performance and Quality of Service (QoS).

This paper addresses this challenge by integrating a variety of allocation algorithms, including random allocation optimized with the simulated annealing algorithm, a Greedy algorithm enhanced with a user-centric heuristic, and a hill-climbing local search algorithm. These methodologies collectively aim to optimize bandwidth distribution and user allocation. Our proposed bandwidth allocation mechanism dynamically adjusts to user demands, ensuring optimal resource distribution and preventing service degradation. This approach establishes a robust and adaptable network environment that outperforms existing models in terms of adaptability, user satisfaction, and operational efficiency [4].

We present a comprehensive (Artificial Intelligence) AI-based framework that employs these algorithms not only as allocation tools but as mechanisms for understanding the dynamics of network resource management too. The Greedy algorithm [5], for instance, prioritizes immediate needs to quickly optimize resource usage. While this method often yields better short-term outcomes by addressing the most urgent requirements first, it inherently lacks foresight, potentially compromising long-term efficiency.

The local search approach, utilizing the Hill Climbing algorithm [6], is particularly effective at balancing load across network slices. By making iterative small adjustments to enhance the current state, this method embodies the principle of incremental improvement. It focuses on immediate gains and on enhancing overall network performance and stability.

Randomized Allocation algorithm [7], underscores the necessity for sophisticated allocation strategies along with it. By indiscriminately assigning network resources, this method highlights the inefficiencies of such randomness and the need for a more strategic allocation.

Simulated Annealing algorithm [8], combined with the randomized allocation, merges the exploratory nature of random allocation with the strategic refinement of simulated annealing. Initially employing a stochastic approach, this algorithm provides a benchmark by using the randomized allocation algorithm. Through the principles of simulated annealing, it iteratively refines this initial allocation using a probabilistic acceptance criterion. This allows the algorithm to escape local optima and explore a broader solution space, balancing exploration and exploitation. Thus, it combines the immediate optimization of random allocation with the strategic refinement and global optimization of simulated annealing, offering a nuanced approach to resource allocation in complex environments.

In the current landscape of 5G network slicing research, various studies have proposed methodologies focusing on IoT, dynamic resource allocation, mechanisms, and mathematical models for resource allocation, among others [9][10][11][12][13]. Prior research has explored heuristic search methods for automated planning and applied heuristic algorithms to solve optimization problems, such as the mapping problem for optimal static allocation of processes on distributed memory architectures. Independent evaluations of hill-climbing and simulated annealing have demonstrated their effectiveness in addressing combinatorial optimization

challenges [14][15]. While these studies provide valuable insights, they often lack a comprehensive approach that integrates user requirements with dynamic, real-time adjustments in network bandwidth allocation. So, there remains a need for approaches that integrate heuristic algorithms with user-centric requirements for network slicing in these environments.

This paper introduces a distinct methodology that considers user-specific requirements for network slicing while employing a multi-algorithmic approach. Central to the methodology is the nature of these algorithms, which align with the dynamic requirements of 5G networks. Service demands within these networks are inherently variable, so this framework is constructed to respond to these variations, thereby optimizing network performance in an ongoing cycle. Unique to this study is the dynamic approach to allocation, which allows the system to adapt to real-time network conditions and user demands. This is particularly relevant in the context of 5G networks, where service demands variable. By employing AI-based algorithms, our framework is designed to continuously improve the network's allocation decisions, ensuring that the network's performance is optimized step by step.

The remainder of the paper is organized as follows. Section II details the operational principles and implementation of three distinct allocation strategies: Randomized Allocation with Simulated Annealing, Greedy, and Local Search with Hill Climbing. Section III describes the setup and specific parameters used to evaluate the AI-based allocation algorithms in a 5G network environment. Section IV provides a comparative analysis of the performance metrics for each algorithm, focusing on their effectiveness in resource distribution and adaptability under varying network loads. The paper concludes with Section V, where we summarize the key findings and discuss potential areas for further research and improvement in network slicing and resource allocation within 5G networks.

II. Allocation Algorithms

The approach taken utilizes three distinct allocation algorithms, each with its unique heuristic designed to optimize the allocation process.

The Greedy Allocation function optimizes resource usage by prioritizing users with higher bandwidth requests. It begins by sorting users in descending order based on their bandwidth requirements, ensuring that those with the most substantial needs are addressed first. Then, it iterates through each user, attempting to allocate them to an available network slice. Within this process, it checks if the user's bandwidth request can be accommodated by the slice's capacity and if allocating the request maintains a positive available bandwidth for the slice. If these conditions are met, the user is added to the slice's user list, and the slice's available bandwidth is adjusted accordingly. This method emphasizes immediate gains by swiftly assigning resources to users with urgent needs.

Algorithm 1 – Greedy Allocation

```
function greedy_allocation(users, slices):
    sort users by bandwidth request in descending order
    for each user in users:
        for each slice in slices:
            if user's bandwidth request is less than or equal to slice's capacity_bandwidth and the available bandwidth after allocating user's request to slice is greater than 0:
                add user to slice's user list
                decrease slice's available_bandwidth by user's bandwidth_request
                print "User <user_id> connected to slice: <slice_id>"
                break out of inner loop
```

The balance ratio calculation algorithm iterates through each slice, calculating the balance as the difference between total and available bandwidth divided by the number of users, and then computes the standard deviation to gauge overall fairness.

Balance Ratio Calculation

```
function calculate_balance_ratio(slices):
    ratios = []
    for each slice in slices:
        if number of users in slice > 0:
            balance = (slice's total_bandwidth - slice's available_bandwidth) / number of users in slice
        else:
            balance = 0
        add balance to ratios
    balance_metric = calculate standard deviation of ratios
    return balance_metric
```

The 'hill_climbing_optimized_for_balance' function extends traditional optimization techniques to prioritize both immediate needs and fair resource distribution. It initializes allocations based on user requests and slice capacities, iteratively refining them to improve balance. By moving users between slices and evaluating the impact on balance, the function aims to achieve a more equitable allocation.

Algorithm 2 – Local Search with Hill Climbing

```
function hill_climbing_optimized_for_balance(slices, users):
    overflow_users = 0
    user_allocation = initialize a dictionary to keep track of which slice each user is allocated to
    for each user in users:
        allocated = False
        for each slice in slices:
            if user's bandwidth request is less than or equal to slice's capacity_bandwidth and user's bandwidth request is less than or equal to slice's available_bandwidth:
                add user to slice's user list
                decrease slice's available_bandwidth by user's bandwidth_request
                update user_allocation dictionary
                allocated = True
                break out of inner loop
        if not allocated:
            print "User <user_id> not allocated because bandwidth request exceeds slice capacities."
            increment overflow_users by 1
    # Optimization for balance
    best_balance_metric = calculate_balance_ratio(slices)
    improved = True
    while improved:
        improved = False
        for each user in users:
            original_slice = user_allocation[user]
            for each slice in slices:
                if slice is not original_slice and user's bandwidth request is less than or equal to slice's capacity_bandwidth and user's bandwidth request is less than or equal to slice's available_bandwidth:
                    # Moving user to a new slice
                    remove user from original_slice's user list
                    increase original_slice's available_bandwidth by user's bandwidth_request
                    add user to slice's user list
                    decrease slice's available_bandwidth by user's bandwidth_request
                    update user_allocation dictionary
                    # Evaluating new balance
                    new_balance_metric = calculate_balance_ratio(slices)
                    if new_balance_metric < best_balance_metric:
                        set improved to True
                        update best_balance_metric to new_balance_metric
                    else:
                        # Reverting the change
                        remove user from slice's user list
                        increase slice's available_bandwidth by user's bandwidth_request
                        add user back to original_slice's user list
```

```
                    decrease original_slice's available_bandwidth by user's bandwidth_request
                    update user_allocation dictionary
                    break out of inner loop
    return overflow_users
```

The provided functions encapsulate a resource allocation strategy within a network environment. The 'Random Allocation' function randomly assigns users to network slices based on their bandwidth and frequency requirements, with a contingency plan for cases where users' needs exceed slice capacities, thereby preventing resource wastage. On the other hand, the 'Simulated Annealing' algorithm optimizes resource allocation iteratively, employing a stochastic approach to explore potential allocations while considering both immediate resource constraints and the broader implications of network balance. The 'Neighbor Generation with Overflow Handling' function plays a crucial role in generating neighboring states for the simulated annealing process, ensuring that any moves adhere to slice capacities and handle overflowed users appropriately. Finally, the 'Cost Calculation' function quantifies the efficiency of a given allocation by assessing overcapacity and the number of overflowed users, providing insights into the effectiveness of the resource allocation strategy.

Algorithm 3 – Simulated Annealing to Optimize Random Search

```
function random_allocation(users, slices):
    for each user in users:
        Shuffle slices randomly
        allocated = False
        for each slice in slices:
            IF (user.bandwidth_request <= slice.capacity_bandwidth) AND (slice.available_hz - user.hz_request > 0):
                Add user to slice.users
                Decrease slice.available_hz by user.hz_request
                allocated = True
                Print "User user.user_id Connected to slice:slice.slice_id"
                break
        if not allocated:
            Print "User user.user_id not allocated because bandwidth request exceeds slice capacities."
            Add user to overflowed_users
function simulated_annealing(slices, users, overflowed_users, initial_temp, cooling_rate, min_temp):
    current_temp = initial_temp
    Randomly allocate users to slices which might create overflowed users
    current_cost = calculate_cost(slices, overflowed_users)
    while current_temp > min_temp:
        next_state, next_overflowed = get_neighbor_with_overflow(slices, users, overflowed_users)
        next_cost = calculate_cost(next_state, next_overflowed)
        cost_diff = next_cost - current_cost
        IF cost_diff < 0 or exp(-cost_diff / current_temp) > random():
            Accept the new state
            slices = next_state
            overflowed_users = next_overflowed
            current_cost = next_cost

        current_temp *= cooling_rate
    return slices, overflowed_users

function get_neighbor_with_overflow(slices, users, overflowed_users):
    Create a shallow copy of slices as new_slices
    potential_users = users + overflowed_users
    user_to_move = random.choice(potential_users)
    current_slice = Find slice where user_to_move is located
    target_slice = Randomly choose a slice from new_slices
    if current_slice != target_slice:
        if current_slice:
            Remove user_to_move from current_slice
            Increase current_slice.available_hz by user_to_move.hz_request

        if target_slice.available_hz >= user_to_move.hz_request:
            Add user_to_move to target_slice
            Decrease target_slice.available_hz by user_to_move.hz_request
            if user_to_move is in overflowed_users:
                Remove user_to_move from overflowed_users
        else:
            if user_to_move was not in any slice:
                Add user_to_move to overflowed_users
    return new_slices, overflowed_users
function calculate_cost(slices, overflowed_users):
    Calculate cost based on over capacity and number of overflowed users
    over_capacity_cost = sum((slice.capacity_bandwidth - slice.available_hz) ^ 2 for slice IN slices IF slice.available_hz < 0)
    overflow_cost = length(overflowed_users) * 100
    return over_capacity_cost + overflow_cost
```

A unique element of this methodology is the dynamic bandwidth reallocation process implemented within the Local Search algorithm. If a user cannot be initially allocated due to all slices being at capacity, the algorithm attempts to redistribute the bandwidth from less utilized slices to accommodate additional users. This process is crucial for enhancing the network's adaptability and overall user satisfaction.

Overflowed Users Reallocation

```
not_allocated = empty list
FOR each overflowed_user IN overflowed:
    PRINT overflowed_user.user_id
    IF slice_configurations['overflow']['total'] - overflowed_user.hz_request >= 0:
        Subtract overflowed_user.hz_request from slice_configurations['overflow']['total']
    ELSE:
        Append overflowed_user.user_id to not_allocated

FOR each not_allocated_user IN not_allocated:
    IF not_allocated_user is not None:
        PRINT not_allocated_user
```

The pseudocode in this algorithm outlines a process to handle overflowed users after a local search hill climb. It iterates through each overflowed user and checks if there's enough remaining bandwidth in the 'overflow' slice to accommodate them. If there is, it deducts their bandwidth request from the total available bandwidth in the 'overflow' slice. If not, it adds the user to the list of not allocated users. This mechanism effectively manages overflowed users using the ‘overflow’ slice.

III. Description of Testbed

The testbed for our simulation is structured around a 5G network environment operated by a macro cell base station with a total spectral capacity of 400MHz. To effectively evaluate AI-based algorithms for optimizing user allocation across network slices, our setup divides this capacity into five distinct slices, each dedicated to different service needs as detailed in Table I. These slices include services ranging from browsing and email with high latency tolerance to ultrahigh-quality video streaming, catering to a broad spectrum of data demands. Each slice is allocated a portion of the total network capacity, ensuring equitable bandwidth distribution.

Our simulation environment is populated with a diverse user base consisting of 250, 400, and 500 users, each requiring bandwidth varying from 1 Mbps to 25 Mbps. The users also experience a wide range of Signal-to-Noise Ratio (SNR) values from 10, indicating subpar conditions, to 45, reflecting excellent connectivity conditions. This setup mimics real-world scenarios where users with varying requirements interact with finite network resources.

TABLE I. SLICE CONFIGURATION FOR EXPERIMENTS

Slice Name	***Description***	***Maximum Throughput***	***Spectrum Allocation***
Browsing and Email	High latency-tolerant applications	Up to 5 Mbps	52 MHz (Slice 0)
VoIP	Voice communications	Up to 1 Mbps	13 MHz (Slice 1)
HDTV	High-definition video content	Up to 16 Mbps	150 MHz (Slice 2)
Video Streaming	Ultrahigh-quality video streaming	Up to 25 Mbps	160 MHz (Slice 3)
Podcasts	Audio streaming services	Up to 2 Mbps	23 MHz (Slice 4)

Each user in the simulation is represented by an instance of the User class, characterized by a unique identifier and a specific bandwidth request (in Mbps) and a request in MHz. The conversion from Mbps to MHz in the context of the Shannon-Hartley theorem involves determining the necessary bandwidth to achieve a specified data rate given a certain signal-to-noise ratio. According to the theorem, the maximum data rate C that can be achieved over a communication channel can be calculated using the formula $C=B\log_2(1+SNR)$, where C is the channel capacity in bits per second, B is the bandwidth in hertz, and SNR is the signal-to-noise ratio in linear terms. This conversion to linear SNR is achieved by the equation $SNR(linear) = 10SNR(dB)/10$. This approach models a realistic scenario where users with diverse data needs connect to a network. This environment along with its associated parameters (the base station in the center and a number of users around it (250, 400, 500) achieving SNR values ranging from 10 to 20 if the user is between the outer circle and the middle circle, 20 to 30 if the user is between the inner circle and the middle circle and 30 to 45 if the user is inside the inner circle) is shown in Figure 1.

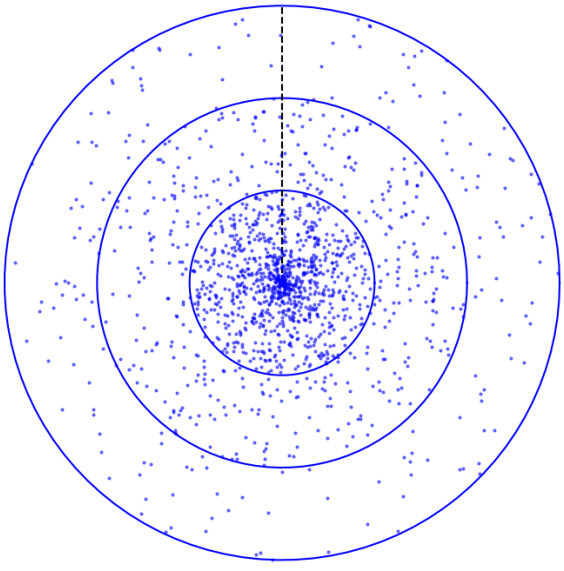

Fig. 1. The Simulation Environment With its Parameters (Base station and Users)

IV. PERFORMANCE EVALUATION

Analyzing the resource allocation across 250, 400, and 500 users in a 5G MIMO network environment provides insightful contrasts between the Greedy, Hill Climbing, and Simulated Annealing algorithms. The disparity in performance across these algorithms underscores the inherent trade-offs between efficiency, complexity, consumption, and overall network utilization. Table II demonstrates that as the number of users decreases, all algorithms perform better, with the Hill Climb and Simulated Annealing algorithms consistently outperforming the Greedy algorithm, particularly at higher user loads. Table III shows that the Simulated Annealing algorithm consistently provides the best balance across all scenarios, followed by the Hill Climb algorithm. The figures 2 through 4 illustrate how each algorithm handles spectrum allocation under different user loads (250, 400, and 500 users).

TABLE II. TOTAL REQUEST MHZ OF OVERFLOWED USERS

Algorithm	***500 Users Scenario***	***400 Users Scenario***	***250 Users Scenario***
Greedy	325 MHz	150 MHz	10 MHz
Hill Climb	260 MHz	130 MHz	0 MHz
Simulated Annealing	250 MHz	120 MHz	0 MHz

TABLE III. BALANCE RATIOS OF THE ALGORITHMS

Algorithm	***500 Users Scenario***	***400 Users Scenario***	***250 Users Scenario***
Greedy	1.5	1.25	1.15
Hill Climb	1.15	1	0.85
Simulated Annealing	0.8	0.7	0.6

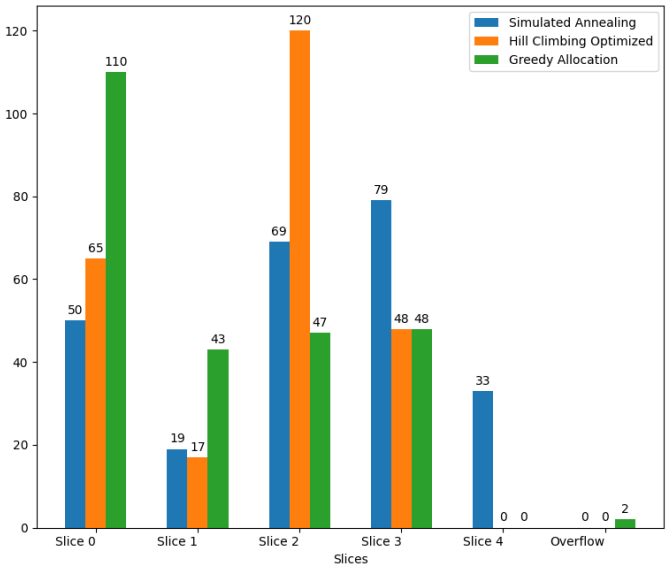

Fig. 2. User Allocation to network slices for 250 users

The figures and tables illustrate the performance of three allocation algorithms—Simulated Annealing, Hill Climbing Optimized, and Greedy Allocation—in distributing users across network slices under varying user densities. Simulated Annealing consistently achieves the most balanced distribution with the lowest number of unsatisfied bandwidth requests and the best balance ratios, indicating optimal performance in managing resources. Hill Climbing also performs well, providing improved balance and fewer unsatisfied requests compared to the Greedy Allocation. The Greedy Allocation algorithm results in the highest number of

unsatisfied requests and the least balanced distribution, particularly under higher user densities.

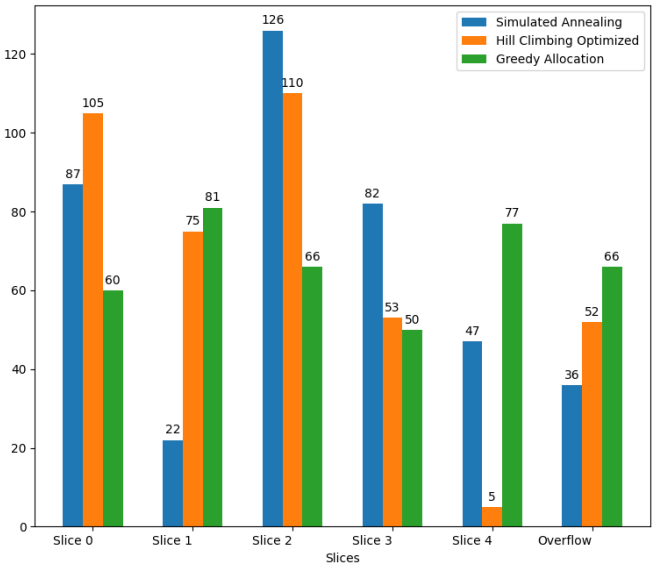

Fig. 3. User Allocation to network slices for 400 users

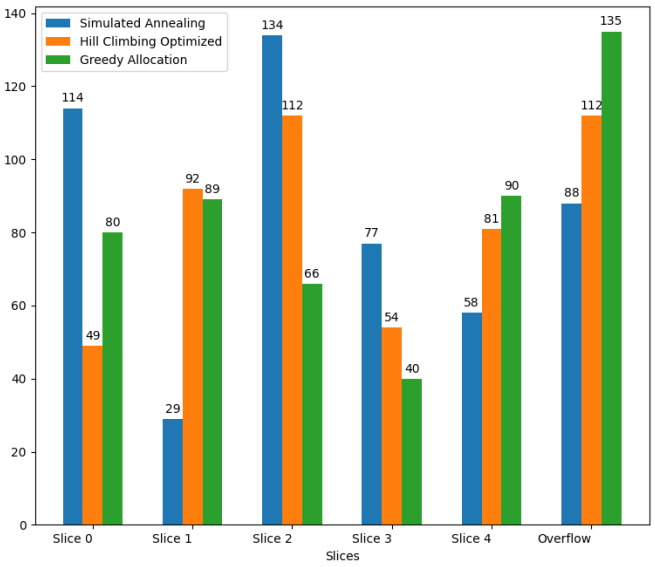

Fig. 4. User Allocation to network slices for 500 users

The Greedy algorithm method is more efficient at fulfilling high bandwidth requests at the outset and is relatively straightforward to implement. It was the least effective in terms of overall resource utilization. With a complexity of O(nlogn+n·m) where n is the number of users, m is the number of slices, it generally enhances user satisfaction by addressing the needs of high-demand users first. However, the greedy nature of the algorithm can also be a disadvantage, as it may lead to suboptimal overall network usage. The Greedy algorithm's focus on immediate gains leads to rapid exhaustion of available resources. As can be seen in tables II, in scenarios with high user densities (400 and 500 users), this approach left the most total unused bandwidth in MHz and did the least efficient resource allocation in terms of balance in slices as observed in the balance ratio score in table III, which calculates the standard deviation of the slices' usage (higher balance ratio means that the slice usage is less balanced). Our findings as shown in Figures 2, 3 and 4 were that early allocations of large requests can exhaust resources quickly and subsequent users with smaller requests either could not be accommodated due to depleted resources or were inefficiently distributed to their second, third or even fourth best slice. The algorithm demonstrated a high balance metric in Table III, reflecting an uneven distribution of resources among network slices. This unevenness indicates that while some slices were heavily utilized, others remained underutilized. This approach proved inefficient in higher density network scenarios, demonstrating that while the Greedy algorithm can be simple and fast, it is sub-optimal in long-term resource distribution.

Local Search algorithm with Hill Climbing builds on the initial allocations made by the greedy algorithm. With a complexity of O(n·m+k·n·m) where n and m are the same as in the greedy algorithm and k is the number of iterations in the hill climbing process. Hill Climbing significantly improved resource utilization by iteratively optimizing the initial allocations. This approach led to a more balanced distribution of resources, reducing the amount of unused bandwidth across slices. So, its adaptive nature becomes useful in scenarios requiring a fair balance in user allocation, as seen in the improved distribution among network slices for all user scenarios in figures 2 through 4. By redistributing resources to achieve better balance, Hill Climbing reduced the number of overflowed users compared to the Greedy algorithm thus resulting in higher user satisfaction. The algorithm also achieved a lower balance metric compared to the Greedy algorithm, indicating a more balanced distribution of resources. This balance is crucial for maintaining consistent network performance and preventing service degradation. However, the trade-off for Hill Climbing is that as the user count increased it required more processing power and time to converge on an optimal or near-optimal allocation solution.

Simulated Annealing algorithm takes a strategic, probabilistic approach to allocation. By allowing for a controlled exploration of allocation possibilities, this algorithm demonstrated superior performance in resource utilization, effectively reallocating resources to minimize unused bandwidth. Its ability to probabilistically accept suboptimal moves enabled it to escape local optima and achieve a more balanced allocation. Simulated Annealing consistently reallocated overflowed users and moved already allocated users around effectively across all scales. Simulated Annealing achieved the lowest overflow rates among the three algorithms which resulted in the highest level of user satisfaction. As seen in the resource allocation Figures 3 and 4, with 400 and 500 users respectively, this algorithm consistently achieved the lowest balance metric, indicating the most equitable distribution of resources among the slices even as the scale increased. It is built to optimize an initial random allocation and together they have a complexity of O(i·(n+m)+n·m) where i is the number of iterations based on the cooling schedule, simulated annealing efficiently found near-optimal solutions, making it suitable for dynamic and high-demand network environments.

The results, as summarized in Tables II and III, highlight the varying degrees of success in meeting user bandwidth requests and utilizing available slice capacity. These indicate that while the Greedy algorithm can quickly allocate resources, it is less effective in meeting user demands and optimizing resource utilization while the other algorithms, with their iterative (Local Search with Hill Climbing) and probabilistic (Simulated Annealing) approaches, provide

superior performance by minimizing unsatisfied user requests and maximizing the utilization of available resources. Simulated Annealing, in particular, consistently shows the best balance between meeting user demands and efficient resource utilization, making it the most effective algorithm for dynamic and high-density network environments. It is understood that each of these algorithms represents a different point on the spectrum of complexity and efficiency and understanding these differences is crucial for implementing the most appropriate resource allocation strategy in 5G networks.

TABLE IV. TIME TAKEN AND MEMORY USAGE

Algorithm	***Time Taken in ms (250/400/500 Users)***	***Memory Usage in KB (250/400/500 Users)***
Greedy	0.011/0.03/0.035	4/4/12
Hill Climb	0.065/0.086/0.11	92/92/92
Simulated Annealing	0.02/0.035/0.045	8/12/12

Regarding the time taken and the memory usage of each of these algorithms, as seen from table IV, because of the strategy each algorithm follows, hill climb took the most time to complete and it is the one that has the worst usage in KB while Greedy search was the cheapest and fastest but cannot account for the reallocation of unsatisfied users.

V. CONCLUSION AND FUTURE WORK

The comparative analysis underscores the importance of selecting an appropriate algorithm based on the specific network characteristics. For 5G environments, where dynamic and efficient resource utilization is crucial, the Local Search with Hill Climbing algorithm holds significant promise. Its sophisticated optimization techniques balance immediate user demands with the goal of equitable network resource distribution. On the other hand, as networks grow in complexity and density, Simulated Annealing stands out for its robust performance. Its approach ensures that even as the scale increases, network efficiency and user satisfaction are not compromised.

Despite the practical applications of the algorithms studied, the theoretical basis for their performance in varying network conditions requires further exploration. For instance, while Simulated Annealing is known for escaping local optima in complex landscapes, its performance can significantly depend on the choice of cooling schedule and temperature parameters.

Further research could explore hybrid approaches that combine the strengths of these algorithms. For instance, combining the predictive capabilities of machine learning with the optimization provided by Simulated Annealing could lead to remarkable advancements in resource allocation.

ACKNOWLEDGMENT

This research has been co-financed by the Hellenic Foundation for Research & Innovation (H.F.R.I) through the H.F.R.I.'s Research Projects to Support Faculty Members & Researchers (project code: 02440).

REFERENCES

[1] X. Foukas, G. Patounas, A. Elmokashfi and M. K. Marina, "Network Slicing in 5G: Survey and Challenges," in IEEE Communications Magazine, vol. 55, no. 5, pp. 94-100, May 2017, doi: 10.1109/MCOM.2017.1600951.

[2] S. Wijethilaka and M. Liyanage, "Survey on Network Slicing for Internet of Things Realization in 5G Networks," in IEEE Communications Surveys & Tutorials, vol. 23, no. 2, pp. 957-994, Secondquarter 2021, doi: 10.1109/COMST.2021.3067807.

[3] M. Jiang, M. Condoluci and T. Mahmoodi, "Network slicing management & prioritization in 5G mobile systems," European Wireless 2016; 22th European Wireless Conference, Oulu, Finland, 2016, pp. 1-6.

[4] R. Su et al., "Resource Allocation for Network Slicing in 5G Telecommunication Networks: A Survey of Principles and Models," in IEEE Network, vol. 33, no. 6, pp. 172-179, Nov.-Dec. 2019, doi: 10.1109/MNET.2019.1900024.

[5] F. Song, J. Li, C. Ma, Y. Zhang, L. Shi and D. N. K. Jayakody, "Dynamic Virtual Resource Allocation for 5G and Beyond Network Slicing," in IEEE Open Journal of Vehicular Technology, vol. 1, pp. 215-226, 2020, doi: 10.1109/OJVT.2020.2990072.

[6] A. A. Abdellatif, A. Mohamed, A. Erbad and M. Guizani, "Dynamic Network Slicing and Resource Allocation for 5G-and-Beyond Networks," 2022 IEEE Wireless Communications and Networking Conference (WCNC), Austin, TX, USA, 2022, pp. 262-267, doi: 10.1109/WCNC51071.2022.9771877.

[7] P. L. Vo, M. N. H. Nguyen, T. A. Le and N. H. Tran, "Slicing the Edge: Resource Allocation for RAN Network Slicing," in IEEE Wireless Communications Letters, vol. 7, no. 6, pp. 970-973, Dec. 2018, doi: 10.1109/LWC.2018.2842189.

[8] C. Campolo, A. Molinaro, A. Iera and F. Menichella, "5G Network Slicing for Vehicle-to-Everything Services," in IEEE Wireless Communications, vol. 24, no. 6, pp. 38-45, Dec. 2017, doi: 10.1109/MWC.2017.1600408.

[9] H. Zhang, N. Liu, X. Chu, K. Long, A. -H. Aghvami and V. C. M. Leung, "Network Slicing Based 5G and Future Mobile Networks: Mobility, Resource Management, and Challenges," in IEEE Communications Magazine, vol. 55, no. 8, pp. 138-145, Aug. 2017, doi: 10.1109/MCOM.2017.1600940.

[10] A. Malik, A. Sharma, Mr. Vinod Saroha (Guide) (2018); Greedy Algorithm; Int J Sci Res Publ 3(8) (ISSN: 2250-3153). http://www.ijsrp.org/research-paper-0813.php?rp=P201564

[11] L. Hernando, A. Mendiburu and J. A. Lozano, "Hill-Climbing Algorithm: Let's Go for a Walk Before Finding the Optimum," 2018 IEEE Congress on Evolutionary Computation (CEC), Rio de Janeiro, Brazil, 2018, pp. 1-7, doi: 10.1109/CEC.2018.8477836.

[12] A. Czumaj and V. Stemann, "Randomized allocation processes," Proceedings 38th Annual Symposium on Foundations of Computer Science, Miami Beach, FL, USA, 1997, pp. 194-203, doi: 10.1109/SFCS.1997.646108.

[13] D. Bertsimas and J. Tsitsiklis, "Simulated annealing," Statistical Science, vol. 8, no. 1, pp. 10-15, Feb. 1993. [Online]. Available: https://doi.org/10.1214/ss/1177011077

[14] Vladimir Ilin, Dragan Simić, Svetislav D Simić, Svetlana Simić, Nenad Saulić, José Luis Calvo-Rolle, A hybrid genetic algorithm, list-based simulated annealing algorithm, and different heuristic algorithms for the travelling salesman problem, Logic Journal of the IGPL, Volume 31, Issue 4, August 2023, Pages 602–617, https://doi.org/10.1093/jigpal/jzac028.

[15] Edmund K. Burke, Yuri Bykov, The late acceptance Hill-Climbing heuristic, European Journal of Operational Research, Volume 258, Issue 1,2017, Pages 70-78, ISSN 0377-2217, https://doi.org/10.1016/j.ejor.2016.07.012.