electronics

MDPI

Article

LRU-GENACO: A Hybrid Cached Data Optimization Based on the Least Used Method Improved Using Ant Colony and Genetic Algorithms

Mulki Indana Zulfa [1,2], Rudy Hartanto [1,*], Adhistya Erna Permanasari [1] and Waleed Ali [3]

[1] Department of Electrical Engineering and Information Technology, Yogyakarta 55281, Indonesia
[2] Department of Electrical Engineering, Jenderal Soedirman University, Purbalingga 53371, Indonesia
[3] Department of Information Technology, King Abdulaziz University, Rabigh 21911, Saudi Arabia
* Correspondence: rudy@ugm.ac.id

Abstract: An optimization strategy for cached data offloading plays a crucial role in the edge network environment. This strategy can improve the performance of edge nodes with limited cache memory to serve data service requests from user terminals. The main challenge that must be solved in optimizing cached data offloading is assessing and selecting the cached data with the highest profit to be stored in the cache memory. Selecting the appropriate cached data can improve the utility of memory space to increase HR and reduce LSR. In this paper, we model the cached data offloading optimization strategy as the classic optimization KP01. The cached data offloading optimization strategy is then improved using a hybrid approach of three algorithms: LRU, ACO, and GA, called LRU-GENACO. The proposed LRU-GENACO was tested using four real proxy log datasets from IRCache. The simulation results show that the proposed LRU-GENACO hit ratio is superior to the LRU GDS SIZE algorithms by 13.1%, 26.96%, 53.78%, and 81.69%, respectively. The proposed LRU-GENACO method also reduces the average latency by 25.27%.

Keywords: cached data offloading; edge network; LRU; ACO; GA

Citation: Zulfa, M.I.; Hartanto, R.; Permanasari, A.E.; Ali, W. LRU-GENACO: A Hybrid Cached Data Optimization Based on the Least Used Method Improved Using Ant Colony and Genetic Algorithms. *Electronics* **2022**, *11*, 2978. https://doi.org/10.3390/electronics11192978

Academic Editor: Neal N. Xiong

Received: 31 August 2022
Accepted: 18 September 2022
Published: 20 September 2022

1. Introduction

The development of Internet of Things (IoT) technology, smart cities, and wearable devices requires reliable edge and cloud network infrastructure [1–3]. Especially in the last two years, during the pandemic, the number of internet users has risen dramatically [4,5]. In addition, the development of 5G networks has resulted in data transactions on cloud networks to be faster and able to send larger packets of data [6,7]. The edge and cloud networks must be able to maintain Quality of Service in controlling latency and reducing network congestion [8,9]. The latency ratio can be reduced as the hit ratio increases. The hit ratio compares data service requests served by the server cache to all data service requests received. A high hit ratio can be achieved by optimizing cached data offloading on each edge node because of its limited storage capacity [10]. Not all data transactions can be cached at each edge node [11].

Cached data offloading helps reduce the number of identical data requests from the edge network to the data center in the cloud network. This strategy faces the challenge of selecting some cached data from among the many accessible candidates [12]. If the strategy is successfully implemented, the computational load on the cloud network will be reduced so it can reduce latency and save bandwidth utilization [13,14]. We use the perspective of Knapsack Problem 0/1 (KP01) to model the cached data offloading problem. KP01 seeks to incorporate as much cached data as possible in cache memory with a limited capacity to generate the most significant profit [15,16].

KP01 is commonly used to tackle traditional optimization problems to maximize profit [17,18]. The profitability of KP01 is dependent on several functions whose minimal

values are sought [11,15]. KP01 is a combinatorial non-polynomial (NP)-hard problem with numerous complex mathematical solutions [19,20]. Therefore, population-based algorithms and swarm intelligence are frequently utilized to solve optimization issues with acceptable solutions. The Ant Colony Optimization (ACO) and Genetic Algorithm (GA) effectively solve the KP01 problem [18,21]. GA has the disadvantage of randomly generated initial solutions but has advantages due to its execution speed [22]. In contrast, ACO requires a longer execution time but has a better average solution than GA [22]. Therefore, the hybrid ACO-GA approach is a reasonable solution because it incorporates the advantages of each of these algorithms [11,15].

This paper continues our previous work on the GENACO framework [11,15] because it does not yet have a handle on cache pollution. Cache pollution can reduce the utility of cache memory. LRU is a caching algorithm that can overcome the problem of cache pollution with a reasonably good hit ratio performance [12,23]. Therefore, this paper contributes to the proposed hybrid LRU-GENACO method, a combination of the three algorithms Least Recently Used (LRU), ACO, and GA. The LRU algorithm is assigned to solve cache-pollution problems, while hybrid ACO-GA optimizes cached data selection before those data are eliminated in cache memory. The proposed method is tested using real proxy log datasets, and then the hit ratio (HR) and latency saving ratio (LSR) are calculated.

The Section 2 of this paper describes similar studies related to cached data offloading strategies. The Section 3 provides an overview of the GENACO framework, a discussion of cache pollution, and an explanation of the concept of the proposed LRU-GENACO method. The Section 4 describes the simulation setup and a description of the dataset used in this paper. The Section 5 describes the discussion of the simulation results. The Section 6 summarizes the paper's conclusions.

2. Related Work

Several intelligent techniques, such as weighting, machine learning (ML), and optimization methods, have been proposed for use in caching strategies by many researchers [11,24,25]. Each of these methods has succeeded in outperforming the HR achievements of several conventional caching algorithms, such as LRU, Least Frequency Used (LFU), and Greedy Dual-Size (GDS). The mechanism for employing weights derived from the statistical data GET/PUT in [26] as a cost variable is still overly generic. The method suggested in [27] can be further optimized at the application level. Several different weight estimates have been proposed, including the usage of network bandwidth consumption [28] and geo-distance [29], which can lead to inaccuracies in an unstable internet network [27].

In addition, numerous researchers have proposed several ML techniques. The ML method in the caching strategy generally requires input in the form of a proxy log to obtain information about the cached data pattern based on recency, access time, access count, data size, or corresponding time. These variables were then analyzed using several ML algorithms to perform grouping based on specific classes or clusters, namely, fuzzy c-means (FCM) with Euclidean distance [26]; FCM with Waterman distance [30]; SVM [29]; decision tree [31]; k-means on cloudlet caching [32]; k-means with fuzzy bi-clustering [33]; naïve Bayes (NB) cooperating with GDS, LRU, and Dynamic Aging (DA) [34]; NB-LRU [35]; J48 [36]; and KNN [37]. Some of these ML algorithms have also been combined with conventional caching algorithms. However, research on caching strategies using the ML method requires additional time and enormous computational resources, particularly during the data-training phase [25,38].

Other studies have compared greedy, branch, and bound heuristic methods, dynamic programming, and metaheuristic methods to tackle data offloading using the KP01 model [39,40]. The greedy algorithm performs quickly but cannot guarantee an optimal solution. The branch-and-bound method offers the optimum solution but requires a long computation time.

KP01's perspective on cached data offloading has a research essence similar to feature-selection research. KP01 aims to select several items from the set of items so as to obtain the most optimum profit. In contrast, feature selection aims to select only the most important features from the many available features to obtain the optimal accuracy. Both of these research problems rely heavily on heuristic functions to choose the most optimal solution. Some researchers also use the ACO algorithm to solve the feature-selection problem. Ant-MCDM [41] used the ACO algorithm with two heuristic functions so that the ACO ants get more information before choosing the optimal feature. Ant-TD [42] utilized reinforcement learning to dynamically assign heuristic functions to select multi-label features. MLACO [43] used unsupervised and supervised learning techniques simultaneously to select the ACO heuristic function on multi-label feature selection. These studies only rely on a single ACO from the first iteration to the maximum iteration. The ACO algorithm has the biggest weakness in execution time because it requires updating the pheromone value by each ant in each iteration. Therefore, the number of ants and the iterations used greatly affect the performance of the ACO algorithm.

Generally, the performance of the heuristic method can be improved using the meta-heuristic method based on population intelligence, as has been done by the cyclic ACO-GA method [11]. We have modified the method in [11] through the GENACO framework [15] so that we were able to find a better solution. This paper continues our previous work on the GENACO framework [15] by adding capabilities to address cache-pollution issues. Cache pollution can reduce cache memory utility and reduce HR performance. Therefore, in this paper, we modified the GENACO framework and combined it with the LRU algorithm to overcome the cache-pollution problem and enhance HR performance.

3. Methodology

3.1. GENACO Framework

Our prior work on the GENACO framework [15] successfully designed the KP01 model to address cached data offloading using noncyclic hybrid ACO-GA. The value $x = 1$ on an object x_i indicates that the *cached data*$_{(i)}$ is included in the knapsack (cache memory). The generic form of KP01 is given by Equation (1). GENACO solves cached data offloading problems using three variables: access count (ct), access time (tm), and data size (sz), and their weights (w_{ct}, w_{tm}, w_{sz}) [11,15]. The three variables are then used to calculate the value of the objective function (Fx) following Equation (2), as well as the profit knapsack (S), whose value was minimized.

The initial population generated by the ACO can be controlled according to its objective target. Each ACO ant used Equation (3) to select the cached data candidates to be entered into the knapsack (cache memory) based on the cumulative probability P. The probability P is influenced by the pheromone value (τ_i^{α}) and the visibility (η_i^{β}) of each cached data x_i entered into the solution set M_i. The ACO algorithm is placed in the first iteration to cover the weakness of the GA, which generates a random initial population. The GA will be run in the 2nd iteration until the maximum iteration after the initial population is formed. The GA algorithm is assigned to exploit the global optimum solution.

$$\sum_{i=1}^{m} profit(F_x)\ x_i \leq S \tag{1}$$

$$F_x = w_{ct} * \left(1 - F_{(ct)}\right) + w_{tm} * \left(1 - F_{(tm)}\right) + w_{sz} * F_{(sz)} \tag{2}$$

$$P_i(t) = \begin{cases} \frac{\tau_i^{\alpha}(t) * \eta_i^{\beta}(t)}{\sum \tau_i^{\alpha}(t) * \eta_i^{\beta}(t)} & if\ \ x_i\ \epsilon\ M_i \\ 0 & other \end{cases} \tag{3}$$

3.2. Discussion of Cache Pollution

LRU is a conventional caching algorithm commonly used by proxy servers in the cache replacement mechanism. LRU has advantages in its ease of implementation and fast performance [35]. LRU is also able to protect the cache memory utility from cache pollution. The LFU and greedy dual-size frequency (GDSF) caching algorithms construct the access count property as a significant consideration in the cache replacement policy. Thus, data with a high access count will be guaranteed to stay longer in cache memory and are quite challenging to replace with other cached data. Figure 1 illustrates how LRU (a) and LFU (b) work in the cache replacement mechanism.

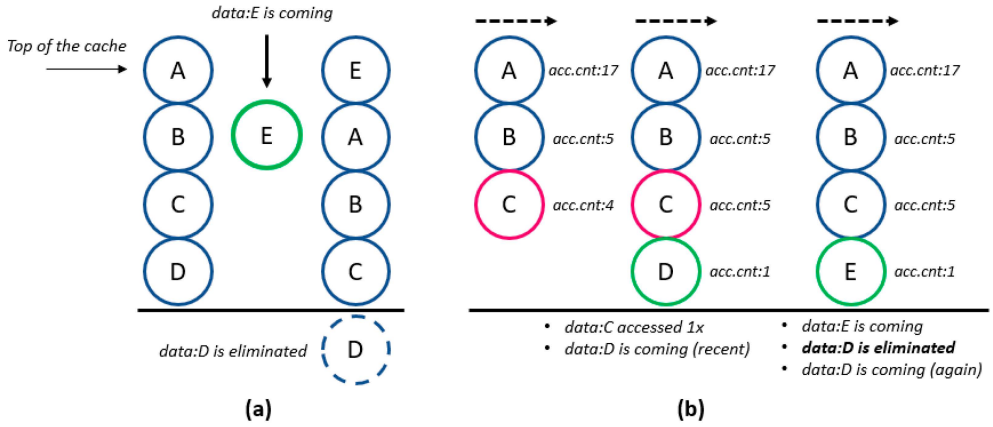

Figure 1. (**a**) LRU algorithm concept; and (**b**) how the LFU works.

The LRU strategy will remove the most recently accessed cached data, as seen in Figure 1a. For instance, new data E, which has been entered into cache memory, will be positioned at the top. The position of data E in Figure 1a will gradually fall to the bottom and subsequently be replaced by other cached data if these data are no longer periodically accessed. According to Figure 1a, data C is automatically relocated at the top of the cache memory if it is accessed at (t + 1).

Cache pollution can be illustrated according to Figure 1b. To identify which cached data to remove, the LFU algorithm relies primarily on access counts. However, suppose the caching algorithm only relies on the access count under certain conditions. In this instance, it will be difficult to replace cached data due to its high cumulative access count value. This condition is exemplified as data A in Figure 1b. According to Figure 1b, when new data D enters the cache memory, these data are immediately placed at the bottom of the cache because it only has one access count. Then, when data E is entered, the presence of data E will delete the data D that was just entered. Even though it could be that in a time interval that is not too long, data D will be reassessed. If this occurs, the cache memory utility will decrease because it cannot use access recency to increase the hit ratio.

$$K_{(g)} = L + F_{(g)} * \frac{C_{(g)}}{S_{(g)}} \tag{4}$$

The weakness of the LRU and LFU has been corrected by expanding the dynamic-aging factor (L) as used by GDSF. The GDSF algorithm considers access frequency $F_{(g)}$, cost data $C_{(g)}$, and data size $S_{(g)}$ into a single key-value, $K_{(g)}$, following Equation (4). GDSF will delete the cached data with the smallest $K_{(g)}$. New cached data that are entered instead of removed data will obtain an additional value of $K_{(g)}$ from the removed data as an L factor. Through this mechanism, the superiority of the cache pollution can be defeated and replaced by new cached data. However, the dynamically aging L offered by GDSF takes a long time to overcome this cache-pollution problem. Cache pollution caused by very popular data (massive or viral access) will be difficult to remove soon using GDSF.

3.3. The Proposed LRU-GENACO Method

The proposed LRU-GENACO method incorporates the LRU and the GDSF algorithm benefits. The GDSF algorithm is superior to conventional caching algorithms because it can formulate access count, data size, and cost in the caching decision process. The performance of GDSF is still very competitive with some new caching methods in obtaining HR values [24,28]. The advantages of the GDSF have been accommodated by the GENACO optimization framework, which is part of the proposed LRU-GENACO method. Equation (2) shows that GENACO considers three cached data property values, the same as those used by the GDSF algorithm. However, GENACO has an advantage in configuring cached data weights that can be adjusted according to the data offloading target to be achieved. We have targeted GENACO to select cached data with a high access count but not a large data size [15].

GDSF can overcome that the already extensive cache pollution still takes a long time because it only relies on the key value from the removed data as the L factor of new data. Therefore, we propose LRU-GENACO as a hybrid method of Ant Colony, Genetic Algorithm (GENACO), and LRU, to be a cached data offloading solution that can overcome the issue of cache pollution to increase the chances of obtaining optimal HR. Figure 2 illustrates how the proposed LRU-GENACO method works.

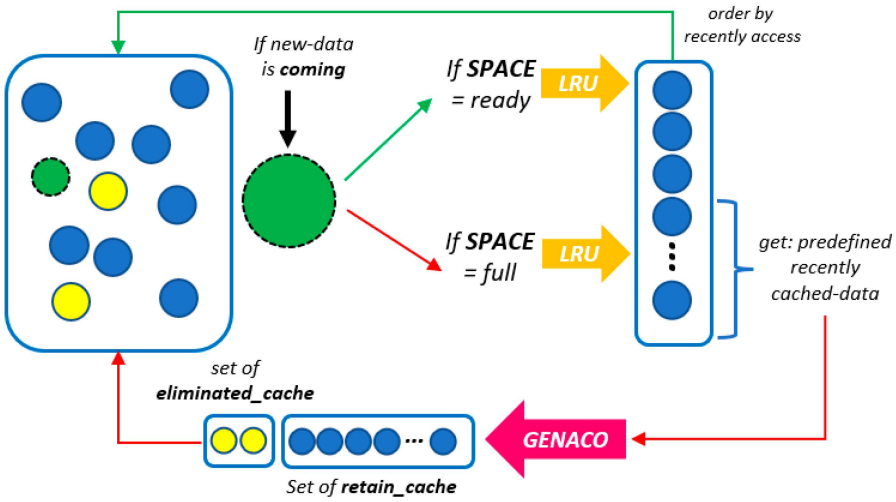

Figure 2. The proposed LRU-GENACO method.

The proposed LRU-GENACO method adopts the working principle of LRU in the cache-replacement mechanism. Every new cached data stored in the cache memory will be sorted by access recency. Such a concept is done to minimize cache pollution. If new cached data arrives and the total cached data in the cache memory are less than the predefined number of cached data, LRU-GENACO runs the LRU policy immediately. That situation is done because of considerations of time efficiency. The proposed LRU-GENACO method will only execute the GENACO optimization framework if the cumulative cached data in the cache memory are more than (1) the predefined, recently used cached data; or (2) a certain cached data percentage threshold. The more the cached data offloading handled by GENACO, the greater the time required. This is a major consideration because GENACO requires additional time compared to the greedy-based conventional caching algorithm. Therefore, the execution of GENACO is restricted to circumstances. GENACO will choose proportionally based on access count (ct), visiting time (tm), and data size (sz). After the cached data offloading optimization by GENACO is completed, two datasets will be formed: (a) a set of eliminated_cache; and (b) a set of retain_cache. Cached data included in the eliminated_cache set will be removed from the cache memory, while those included in the retain_cache set will be retained. It can be concluded that the proposed LRU-GENACO method has two decision options, namely, (i) directly running LRU (green-line); or (2)

running GENACO optimization if certain conditions have been met (red-line), as shown in Figure 2. Algorithm 1 presents the pseudo-code of the proposed LRU-GENACO method.

Algorithm 1 Pseudo-code the proposed LRU-GENACO method

Input: GENACO, LRU, n new cached_data
Output: set of eliminated_cache, set of retain_cache

```
if n is coming then
  check cache_archive(n);
  if isReady(n) then
    # cache hits
    update(stat(n)); update(info(ct,tm,sz));
  else
    # cache miss and space ready
    if size(n) < remaining_cache_capacity then
      insert(n); update(stat(ct)); update(info(ct,tm,sz));
    else
      # cache miss and space full, do optimization
      if sum_cache < 20 then
        set_n = all cached data
        run(LRU,n); remove(set_em);
      else
        set_n = get(20 recently used cached data)
        run(GENACO,set_n);
        screen_out(set_ren); remove(set_em);
        insert(n); update(stat(ct)); update(info(ct,tm,sz));
      endif
    endif
  endif
endif
```

4. Simulation and Datasets

In our prior work, noncyclic ACO-GA (GENACO) [15] outperformed cyclic ACO-GA (CGACA) [11] in terms of an optimal objective function value. In this paper, the proposed LRU-GENACO method is tested more comprehensively using four real proxy-log datasets from IRCache used in [25,35,44]. Subsequently, we tested it by calculating the hit ratio.

4.1. Performance Metrics

Calculating the hit ratio is the most frequently used way by researchers to measure the performance of the proposed caching strategy [12,45,46]. The hit ratio is the percentage (0–100%) between data service requests that was successfully served by the server cache d_i compared to the total data service requests received by the server (N). At the same time, t_i is the time it takes to download $data_{(i)}$ from the origin server to the server cache. The higher the HR generated, the greater the latency that can be reduced [47,48]. In addition to HR measurement, this study also includes LSR performance measurement results. LSR is defined as the ratio of the sum of the download times of objects satisfied by the cache over the sum of all downloading times [12,45]. HR and LSR are calculated using Equations (5) and (6), respectively.

$$HR = \frac{\sum_{i=1}^{N} d_i}{N} \tag{5}$$

$$LSR = \frac{\sum_{i=1}^{N} t_i d_i}{\sum_{i=1}^{N} t_i} \tag{6}$$

4.2. Datasets

We used four separate datasets obtained from IRCache. These four datasets have different information properties from each other. To simplify the simulation, we only used 666 records from each dataset. Table 1 describes the four datasets used in the simulations in this paper. Based on Table 1, we can understand that the NY dataset has the best chance of achieving HR since the NY dataset has the smallest requests of cacheable and unique data. However, the SV dataset will be the smallest HR achievement regarding the ratio between cacheable and unique requests. Therefore, these four datasets can represent data-access patterns in the real environmental conditions of edge networks.

Table 1. Description of the four datasets used in this paper.

No	Info	BO2	SV	UC	NY
1	**Proxy Location**	Boulder, Colorado	Silicon Valley, California	Urbana Champaign	New York
2	**Total requests**	666	666	666	666
3	**Cacheable requests**	110	87	141	74
4	**Cacheable bytes**	57,629	44,480	71,583	37,218
5	**Unique requests**	386	527	388	267

5. Result and Discussion

In this paper, the main performance measure was calculated using the hit ratio. However, we also calculated how much the latency ratio can be reduced based on the proposed LRU-GENACO method.

5.1. Impact of Cache Size on HR Performance

Based on Figure 3, the size of the cache memory has a significant impact on HR performance. The larger the cache memory size is, the higher the HR that can be achieved. The HR increment rate decreases as the simulation approaches the maximum cache memory size. All algorithms obtain the maximum HR at three cache memory sizes: 200, 250, and 300 (KB). Such a condition shows that all data requests on the 666 lines of the proxy log can be served entirely by the server cache. The data offloading process is often carried out on a small cache memory size. In very limited cache memory space, all caching algorithms struggle to store popular cached data. This has an impact on the achievement of a relatively low HR score. However, this small cache memory capacity has a positive impact because there will be no cache-pollution problems.

Simulations on the BO2 dataset show that the HR achieved by LRU-GENACO is superior in seven cache size configurations: 3, 6, 9, 12, 15, 20, and 25 (KB). The SIZE algorithm is the worst in the simulation of the BO2 and UC datasets. The SIZE caching algorithm will fill the cache memory with ranging from small data to full cache memory. However, repeated data access requests on the BO2 or UC datasets do not constantly occur in small cached data. Therefore, even though the SIZE algorithm can store more cached data, it cannot increase HR performance.

According to Figure 3, the LFU algorithm records the worst HR in the SV dataset. The LFU algorithm makes rapid decisions to clear cached data with a small access count. At the same time, access to cached data occurs repeatedly but not at the same time. The GDS algorithm obtains the lowest HR among the algorithms in the NY dataset. The GDS algorithm has fixed the cache-pollution problem in the SIZE algorithm. Small cached data in the SIZE algorithm are difficult to replace with other cached data. Therefore, the GDS algorithm adds an L factor to reduce this cache-pollution problem. However, repeated data requests on the NY dataset do not constantly occur in small cached data. Therefore, the GDS algorithm becomes the worst in the NY dataset simulation.

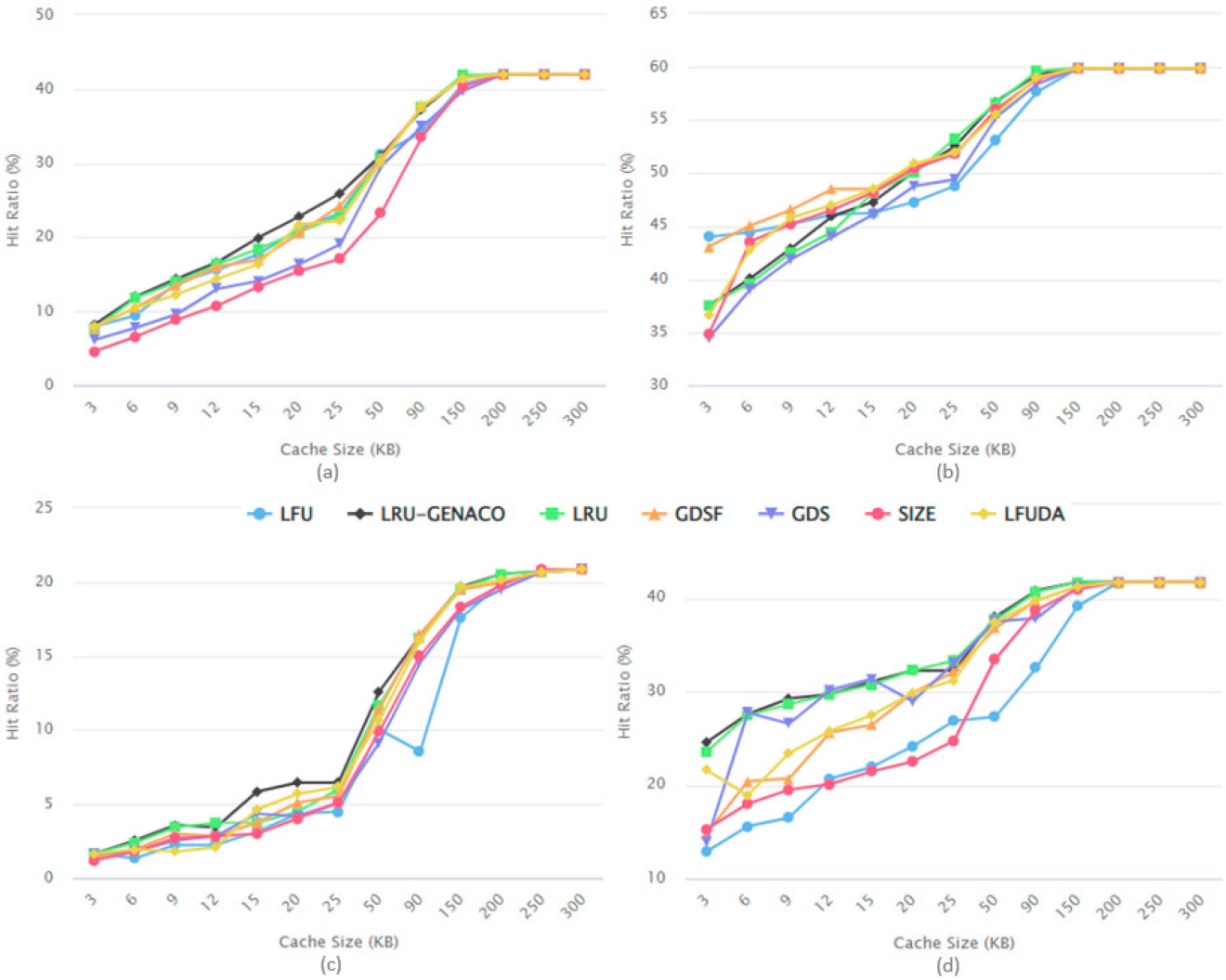

Figure 3. Impact of cache size on HR performance of four different datasets: (**a**) BO2; (**b**) NY; (**c**) SV; and (**d**) UC.

Based on Table 2, the proposed LRU-GENACO method has the best average HR achievement from the simulation of four IRCache datasets. LRU and GDSF achieved the second- and third-best average HR achievements. The LFU algorithm is the worst in terms of average HR performance. In theory, the LFU algorithm will store more popular cached information. Therefore, it should increase the chances of increasing HR performance. However, the data offloading problem has other constraints on cache memory capacity and cached-data size. The access count aspect cannot be used as the only factor for making caching decisions. The recency factor and the size of the cached data must be considered when making caching decisions.

Table 2. HR average comparison over conventional caching algorithms.

No	Datasets	HR Average						
		LRU	LFU	LFUDA	SIZE	GDS	GDSF	LRU-GENACO
1	BO2	26.78	26.25	26.23	23.1	24.36	26.59	**27.39**
2	NY	51.65	51.73	52.14	51.93	50.54	**52.97**	51.71
3	SV	10.39	9.02	10.16	9.66	9.65	10.2	**10.83**
4	UC	34.71	27.93	32.47	29.24	33.37	31.79	**34.82**
AVG		30.88	28.73	30.25	28.48	29.48	30.39	**31.19**

The proposed LRU-GENACO method was developed based on the working principle of LRU, so in the NY dataset simulation, both have poor performance. However, LRU-GENACO's performance does not differ much from the other five caching algorithms. Based on Table 2, the GDSF algorithm has the best performance on the NY dataset simulation. The NY dataset has a unique characteristic: there are many simultaneous accesses to cached data in a specific time; it disappears for a moment, and then the simultaneous access comes back.

5.2. Reduced Latency Ratio by LRU-GENACO

The more significant the HR is, the faster data requests can be sent to users because the data requests are served directly by cache memory on edge nodes closer to end users. Table 3 compares the LSR by the LRU-GENACO method against the conventional caching method. The LSR value is directly proportional to the available cache memory configuration. The larger the cache memory is, the greater the latency ratio that can be reduced. Table 3 also shows that the implementation of the caching strategy can significantly reduce the latency that occurs in the server environment, which also affects bandwidth efficiency and reduces the occurrence of network congestion.

Table 3. LSR average comparison over conventional caching algorithms.

No	Datasets	Average LSR						
		LRU	**LFU**	**LFUDA**	**SIZE**	**GDS**	**GDSF**	**LRU-GENACO**
1	BO2	18.99	11.38	16.29	13.75	8.22	12.56	**22.07**
2	SV	1.31	1.05	1.8	1.45	1.39	1.96	**3.13**
3	UC	17.86	5.86	16.422	5.41	8.72	13.99	**18.17**
4	NY	24.56	23.97	26.08	25.9	25.97	**26.52**	25.27
AVG		15.68	10.57	15.15	11.63	11.08	13.76	**17.16**

Based on Table 1, the NY dataset has the least number of cacheable requests compared to other datasets, so the data offloading mechanism in this dataset is also minimal. It has the most significant impact on HR and LSR values. Based on Table 3, the latency ratio that can be reduced from the NY dataset is more than 25%. In contrast, because the SV dataset has the highest number of cacheable requests, the maximum latency ratio can be reduced by only 3.13%. The proposed LRU-GENACO method has reduced the most significant latency ratio among other caching algorithms. The proposed LRU-GENACO method is superior to 62% of the average LSR generated by LFU. The average LSR achieved by the proposed LRU-GENACO method excels from the best caching algorithm, GDSF, by 25%.

Based on our direct observations, the same cached data exist but with slightly different download times. Therefore, the LSR measurement can result in bias because the latency in actual conditions is influenced by bandwidth conditions and the internet network topology [27]. However, the simulation results show that the HR value is directly proportional to the LSR value. The latency ratio is affected by the amount of cached data that can be stored and the configuration of the available cache memory. Although the two caching methods have the same HR value, the achievement of the LSR value can be different. Simulations on the UC dataset show that the GDS and GDSF algorithms have the same HR value of 17.87% in the 3 KB cache size, but the resulting LSRs are different: 15.4% and 7.44%, respectively. The caching approach that maintains more extensive cached data has a better LSR value at the same HR value.

5.3. Improvement Ratio by LRU-GENACO

We have tested the performance of the proposed LRU-GENACO method using four IRCache datasets by calculating HR and LSR. In general, the achievement of the LSR is directly proportional to the HR value. A higher value of both indicates that the cached data offloading mechanism in the caching strategy is more effective in reducing the server

workload. We calculated the improvement ratio to show the positive contribution of the proposed LRU-GENACO method compared to other caching algorithms. The improvement ratio (IR) is the ratio of the increase in HR value achieved by the proposed method (Pm) to the conventional method (Cm), which is calculated using Equation (7). A negative sign ("-") indicates that Pm performs worse than Cm. IR is only calculated on the size of the cache memory, which has a distinct HR value. We did not calculate IR on the last three cache memory sizes—200, 250, and 300 (KB)—because all the caching algorithms produce the same HR for these configurations.

Table 4 shows the improvement ratio (IR) calculation exemplified in the BO2 dataset. The proposed LRU-GENACO method has increased the HR performance from the conventional LRU algorithm by 13.1%. Then, the GENACO optimization framework, which is part of the proposed method, increased the HR performance from the conventional GDSF algorithm by 17.68%. The HR improvement from the proposed LRU-GENACO method is the largest at 81.69% compared to the SIZE caching algorithm.

$$IR = \frac{P_m - C_m}{C_m} * 100(\%) \quad (7)$$

Table 4. Improvement ratio by LRU-GENACO for the six caching algorithms.

No	Cache Size	Hit Ratio (HR)					
		LRU	**LFU**	**LFUDA**	**SIZE**	**GDS**	**GDSF**
1	3000	**7.83**	**3.77**	**3.77**	**77.63**	**34.09**	**5.76**
2	6000	**1.26**	**26.96**	**14.27**	**81.69**	**53.78**	**14.27**
3	9000	**3.22**	**4.34**	**17.06**	**62.64**	**49.95**	**6.66**
4	12,000	**1.83**	**6.72**	**15.68**	**54.21**	**27.64**	**3.73**
5	15,000	**8.12**	**12.7**	**21.99**	**49.48**	**41.53**	**17.68**
6	20,000	**10.14**	**7.79**	**4.82**	**47.51**	**39.4**	**10.94**
7	25,000	**13.1**	**11.65**	**16.92**	**51.75**	**36.23**	**6.83**
8	50,000	**1.47**	−0.48	**2.98**	**33.56**	**5.07**	**1.47**
9	90,000	−0.8	**8.79**	−1.19	**10.73**	**6.46**	−1.19
10	150,000	−1	**1.92**	**0.07**	**2.67**	**4.22**	**0.07**

5.4. Discussion

The previous section showed that the proposed LRU-GENACO method is superior to the three datasets BO2, SV, and UC. The simulation on the NY dataset shows a slightly decreased performance because this dataset has a different access character than the general dataset. Although the performance of LRU-GENACO on the NY dataset seems to be declining, the IR calculations on the proposed LRU-GENACO method have successfully contributed 22 times out of 54 times to the IR measurements (60% contributed positively). We understand that the NY dataset has different access characteristics from the BO2, SV, and UC datasets. We searched for the data access patterns of the four datasets and then calculated the cumulative data access (Cnt), average (Avg), and standard deviation (STDev). The results show an anomaly in the NY dataset, which has an STDev = 23.23, which is far from the Avg = 3.64.

Such points are more beneficial for GDSF because GDSF is more relevant to large amounts of data access. Therefore, HR GDSF in the NY dataset is more significant than LRU-GENACO. However, the proposed LRU-GENACO technique is more cautious than GDSF in dealing with this abnormality. Mistakes in caching decisions can lead to a decrease in HR value in the future. Our subsequent work will develop an adaptive LRU-GENACO capable of understanding the data access anomaly.

Cached data offloading optimization combined with the LRU algorithm used by LRU-GENACO is an advantage shown in this paper. The simulation results show that LRU-GENACO is superior to the six conventional caching algorithms in terms of HR and

LSR. Future work that can be developed is to combine GENACO with other conventional caching algorithms. However, combining these methods may result in additional time.

The more particles and iterations run, the larger the execution time of LRU-GENACO. The number of particles and iterations can be determined based on the characteristics of the handled cached data. The GENACO optimization framework is set to use ten particles in ACO and GA, with a maximum of 20 iterations. The performance of ACO is determined by the heuristic function (pheromone, τ) and its visibility (η). The heuristic function plays a significant role in exploiting. At the same time, visibility can explore the search space as wide as possible to avoid local optimum solutions that can reduce the HR values.

Reducing the number of particles and iterations can speed up the execution time of LRU-GENACO. Assuming the number of cached data is n, the number of particles is m, and the number of iterations is k, then the GENACO optimization framework complexity is O(k*n*m). Subsequently, the conventional LRU caching algorithm has the most superior complexity, O(1), when compared to LFU O(Log(n)) and GDSF O(nlog2(n)). Therefore, the proposed LRU-GENACO method will have a complexity of O(1) before satisfying the predefined cache or have a complexity of O(k*n*m) if it has to run a cached data optimization mechanism. Relatively high complexity is a weakness of LRU-GENACO, but this method can produce a superior hit ratio to the other conventional caching algorithms.

Meanwhile, the proposed LRU-GENACO method can practically be implemented on a proxy server or database server for managing the same and repeated data access to reduce latency and bandwidth efficiency.

6. Conclusions

This paper proposed the LRU-GENACO method as a hybrid method of three algorithms, LRU and ACO-GA (GENACO), to solve cached data offloading optimization. The proposed LRU-GENACO method has two choices for data offloading decisions: running data offloading using LRU, to achieve HR by access recency; or optimizing using the GENACO framework on predefined cached data, to reduce cache-pollution problems on the cache server. The proposed method was tested using real proxy-log datasets. The simulation results show that LRU-GENACO is superior to the six conventional caching algorithms, as indicated by the improvement ratio (IR), in achieving the hit ratio (HR). This paper also presents the measurement of LSR. Furthermore, HR and LSR values are directly proportional to the increment in cache memory capacity. Our future work could be to develop an adaptive LRU-GENACO capable of capturing insights into its data access patterns before making caching decisions.

Author Contributions: Conceptualization, M.I.Z., R.H., A.E.P. and W.A.; formal analysis, M.I.Z.; investigation, R.H. and A.E.P.; methodology, M.I.Z. and W.A.; project administration, R.H. and A.E.P.; supervision, R.H. and A.E.P.; writing—original draft, M.I.Z.; writing—review and editing, M.I.Z. and W.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Rekognisi Tugas Akhir (RTA) Grant No. 1525/UN1/DITLIT/Dit-Lit/PT.01.05/2022 from Gadjah Mada University (UGM), and also by LPDP, as scholarship provider through BUDI-DN in the doctoral study program DTETI UGM.

Acknowledgments: The authors thank the supervisory team for their suggestions and motivation. We are also grateful for the reviewers' constructive comments that improved the quality of this paper.

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

References

1. Ha, K.; Chen, Z.; Hu, W.; Richter, W.; Pillai, P.; Satyanarayanan, M. Towards wearable cognitive assistance. In Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services, Bretton Woods, NH, USA, 16–19 June 2014; pp. 68–81. [CrossRef]
2. He, W.; Zhang, Z.; Li, W. Information technology solutions, challenges, and suggestions for tackling the COVID-19 pandemic. *Int. J. Inf. Manag.* **2020**, *57*, 102287. [CrossRef] [PubMed]

3. Secundo, G.; Shams, S.R.; Nucci, F. Digital technologies and collective intelligence for healthcare ecosystem: Optimizing Internet of Things adoption for pandemic management. *J. Bus. Res.* **2021**, *131*, 563–572. [CrossRef]
4. De', R.; Pandey, N.; Pal, A. Impact of digital surge during Covid-19 pandemic: A viewpoint on research and practice. *Int. J. Inf. Manag.* **2020**, *55*, 102171. [CrossRef] [PubMed]
5. Nimrod, G. Changes in Internet Use When Coping WITH Stress: Older Adults during the COVID-19 Pandemic. *Am. J. Geriatr. Psychiatry* **2020**, *28*, 1020–1024. [CrossRef]
6. Azlan, C.A.; Wong, J.H.D.; Tan, L.K.; Huri, M.S.N.A.; Ung, N.M.; Pallath, V.; Tan, C.P.L.; Yeong, C.H.; Ng, K.H. Teaching and learning of postgraduate medical physics using Internet-based e-learning during the COVID-19 pandemic—A case study from Malaysia. *Phys. Med.* **2020**, *80*, 10–16. [CrossRef]
7. Naeem, M.; Ozuem, W. The role of social media in internet banking transition during COVID-19 pandemic: Using multiple methods and sources in qualitative research. *J. Retail. Consum. Serv.* **2021**, *60*, 102483. [CrossRef]
8. Sai, Y.; Fan, D.-Z.; Fan, M.-Y. Cooperative and efficient content caching and distribution mechanism in 5G network. *Comput. Commun.* **2020**, *161*, 183–190. [CrossRef]
9. Ayuba, D.; Ismail, A.; Hamzah, M.I. Evaluation of Page Response Time between Partial and Full Rendering in a Web-based Catalog System. *Procedia Technol.* **2013**, *11*, 807–814. [CrossRef]
10. Carvalho, G.; Cabral, B.; Pereira, V.; Bernardino, J. Computation offloading in Edge Computing environments using Artificial Intelligence techniques. *Eng. Appl. Artif. Intell.* **2020**, *95*, 103840. [CrossRef]
11. Wang, D.; An, X.; Zhou, X.; Lü, X. Data cache optimization model based on cyclic genetic ant colony algorithm in edge computing environment. *Int. J. Distrib. Sens. Netw.* **2019**, *15*, 155014771986786. [CrossRef]
12. Ali, W.; Shamsuddin, S.M.; Ismail, A.S. A Survey of Web Caching and Prefetching. *Int. J. Adv. Soft Comput. Appl.* **2011**, *3*, 1–27.
13. Zulfa, M.I.; Fadli, A.; Wardhana, A.W. Application caching strategy based on in-memory using Redis server to accelerate relational data access. *J. Teknol. Dan Sist. Komput.* **2020**, *8*, 157–163. [CrossRef]
14. Baskaran, K.R.; Kalaiarasan, C. Pre-eminence of Combined Web Pre-fetching and Web Caching Based on Machine Learning Technique. *Arab. J. Sci. Eng.* **2014**, *39*, 7895–7906. [CrossRef]
15. Zulfa, M.I.; Hartanto, R.; Permanasari, A.E.; Ali, W. GenACO a multi-objective cached data offloading optimization based on genetic algorithm and ant colony optimization. *PeerJ Comput. Sci.* **2021**, *7*, e729. [CrossRef]
16. Ying, C.; Wang, X.; Luo, Y. Optimization on data offloading ratio of designed caching in heterogeneous mobile wireless networks. *Inf. Sci.* **2021**, *545*, 663–687. [CrossRef]
17. Shi, H. Solution to 0/1 Knapsack Problem Based on Improved Ant Colony Algorithm. In Proceedings of the 2006 IEEE International Conference on Information Acquisition, Shandong, China, 20–23 August 2006; pp. 1062–1066. [CrossRef]
18. Fidanova, S. Ant Colony Optimization for Multiple Knapsack Problem and Model Bias. *Lect. Notes Comput. Sci.* **2005**, *3401*, 280–287. [CrossRef]
19. Liu, Y.; Gao, C.; Zhang, Z.; Lu, Y.; Chen, S.; Liang, M.; Tao, L. Solving NP-Hard Problems with Physarum-Based Ant Colony System. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **2017**, *14*, 108–120. [CrossRef]
20. Ansari, A.Q.; Ibraheem; Katiyar, S. Comparison and analysis of solving travelling salesman problem using GA, ACO and hybrid of ACO with GA and CS. In Proceedings of the 2015 IEEE Workshop on Computational Intelligence: Theories, Applications and Future Directions (WCI), Kanpur, India, 14–17 December 2015; pp. 1–5. [CrossRef]
21. Lin, F.-T. Solving the knapsack problem with imprecise weight coefficients using genetic algorithms. *Eur. J. Oper. Res.* **2008**, *185*, 133–145. [CrossRef]
22. Zulfa, M.I.; Hartanto, R.; Permanasari, A.E. Performance Comparison of Swarm Intelligence Algorithms for Web Caching Strategy. In Proceedings of the 2021 IEEE International Conference on Communication, Networks and Satellite (COMNETSAT), Online, 17–18 July 2021; pp. 45–51. [CrossRef]
23. Zulfa, M.I.; Hartanto, R.; Permanasari, A.E. Caching strategy for Web application—A systematic literature review. *Int. J. Web Inf. Syst.* **2020**, *16*, 545–569. [CrossRef]
24. Ma, T.; Qu, J.; Shen, W.; Tian, Y.; Al-Dhelaan, A.; Al-Rodhaan, M. Weighted Greedy Dual Size Frequency Based Caching Replacement Algorithm. *IEEE Access* **2018**, *6*, 7214–7223. [CrossRef]
25. Ali, W.; Shamsuddin, S.M.; Ismail, A.S. Intelligent Web proxy caching approaches based on machine learning techniques. *Decis. Support. Syst.* **2012**, *53*, 565–579. [CrossRef]
26. Bengar, D.A.; Ebrahimnejad, A.; Motameni, H.; Golsorkhtabaramiri, M. A page replacement algorithm based on a fuzzy approach to improve cache memory performance. *Soft Comput.* **2020**, *24*, 955–963. [CrossRef]
27. Hou, B.; Chen, F. GDS-LC: A latency-and cost-aware client caching scheme for cloud storage. *ACM Trans. Storage* **2017**, *13*, 1–33. [CrossRef]
28. Ma, T.; Hao, Y.; Shen, W.; Tian, Y.; Al-Rodhaan, M. An Improved Web Cache Replacement Algorithm Based on Weighting and Cost. *IEEE Access* **2018**, *6*, 27010–27017. [CrossRef]
29. Aimtongkham, P.; So-In, C.; Sanguanpong, S. A novel web caching scheme using hybrid least frequently used and support vector machine. In Proceedings of the 2016 13th International Joint Conference on Computer Science and Software Engineering, JCSSE 2016, Khon Kaen, Thailand, 13–15 July 2016; pp. 1–6. [CrossRef]
30. Patel, D. Threshold based partial partitioning fuzzy means clustering algorithm (TPPFMCA) for pattern discovery. *Int. J. Inf. Technol.* **2020**, *12*, 215–222. [CrossRef]

31. Nimishan, S.; Shriparen, S. An Approach to Improve the Performance of Web Proxy Cache Replacement Using Machine Learning Techniques. In Proceedings of the 2018 IEEE 9th International Conference on Information and Automation for Sustainability, ICIAfS 2018, Colombo, Sri Lanka, 21–22 December 2018; pp. 1–6. [CrossRef]
32. Zhang, Z.; Hao, W. Development of a new cloudlet content caching algorithm based on web mining. In Proceedings of the 2018 IEEE 8th Annual Computing and Communication Workshop and Conference, CCWC 2018, Las Vegas, NV, USA, 8–10 January 2018; Volume 2018, pp. 329–335. [CrossRef]
33. Pernabas, J.B.; Fidele, S.F. Enhancements to greedy web proxy caching algorithms using data mining method and weight assignment policy. *Int. J. Innov. Comput. Inf. Control* **2018**, *14*, 1311–1326. [CrossRef]
34. Ali, W.; Shamsuddin, S.M.; Ismail, A.S. Intelligent Naïve Bayes-based approaches for Web proxy caching. *Knowl.-Based Syst.* **2012**, *31*, 162–175. [CrossRef]
35. Ali, W.; Sulaiman, S.; Ahmad, N. Performance improvement of least-recently-used policy in web proxy cache replacement using supervised machine learning. *Int. J. Adv. Soft Comput. Its Appl.* **2014**, *6*, 1–38.
36. Ibrahim, H.; Yasin, W.; Udzir, N.I.; Hamid, N.A.W.A. Intelligent cooperative web caching policies for media objects based on J48 decision tree and Naïve Bayes supervised machine learning algorithms in structured peer-to-peer systems. *J. Inf. Commun. Technol.* **2016**, *15*, 85–116. [CrossRef]
37. Pernabas, J.B.; Fidele, S.F.; Vaithinathan, K.K. Enhancing Greedy Web Proxy caching using Weighted Random Indexing based Data Mining Classifier. *Egypt. Inform. J.* **2019**, *20*, 117–130. [CrossRef]
38. Mertz, J.; Nunes, I. Automation of application-level caching in a seamless way. *Softw. Pract. Exp.* **2018**, *48*, 1218–1237. [CrossRef]
39. Ezugwu, A.E.; Pillay, V.; Hirasen, D.; Sivanarain, K.; Govender, M. A Comparative Study of Meta-Heuristic Optimization Algorithms for 0–1 Knapsack Problem: Some Initial Results. *IEEE Access* **2019**, *7*, 43979–44001. [CrossRef]
40. Rashkovits, R. Preference-based content replacement using recency-latency tradeoff. *World Wide Web* **2016**, *19*, 323–350. [CrossRef]
41. Hashemi, A.; Joodaki, M.; Joodaki, N.Z.; Dowlatshahi, M.B. Ant colony optimization equipped with an ensemble of heuristics through multi-criteria decision making: A case study in ensemble feature selection. *Appl. Soft Comput.* **2022**, *124*, 109046. [CrossRef]
42. Paniri, M.; Dowlatshahi, M.B.; Nezamabadi-Pour, H. Ant-TD: Ant colony optimization plus temporal difference reinforcement learning for multi-label feature selection. *Swarm Evol. Comput.* **2021**, *64*, 100892. [CrossRef]
43. Paniri, M.; Dowlatshahi, M.B.; Nezamabadi-Pour, H. MLACO: A multi-label feature selection algorithm based on ant colony optimization. *Knowl.-Based Syst.* **2020**, *192*, 105285. [CrossRef]
44. Ali, W.; Shamsuddin, S.M. Intelligent Dynamic Aging Approaches in Web Proxy Cache Replacement. *J. Intell. Learn. Syst. Appl.* **2015**, *7*, 117–127. [CrossRef]
45. Mertz, J.; Nunes, I. Understanding Application-Level Caching in Web Applications. *ACM Comput. Surv.* **2017**, *50*, 1–34. [CrossRef]
46. Chen, T.-H.; Shang, W.; Hassan, A.E.; Nasser, M.; Flora, P. CacheOptimizer: Helping developers configure caching frameworks for hibernate-based database-centric web applications. In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Seattle, WA, USA, 13–18 November 2016; pp. 666–677. [CrossRef]
47. Kroeger, T.M.; Long, D.D.E. Exploring the Bounds of Web Latency Reduction from Caching and Prefetching. In Symposium on Internet Technologies and Systems on USENIX, 1997, no. September 2012, [Online]. Available online: https://dl.acm.org/citation.cfm?id=1267281 (accessed on 29 September 2020).
48. Teng, W.-G.; Chang, C.-Y.; Chen, M.-S. Integrating Web caching and Web prefetching in client-side proxies. *IEEE Trans. Parallel Distrib. Syst.* **2005**, *16*, 444–455. [CrossRef]