I. J. Education and Management Engineering, **2023, 1, 10-17**
Published Online on February 8, 2023 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijeme.2023.01.02

Optimal and Appropriate Job Allocation Algorithm for Skilled Agents under a Server Constraint

Mijanur Rahaman
Department of Computer Science & Engineering, Bangladesh University of Business & Technology, Bangladesh
E-mail: riponcs.it@bubt.edu.bd

Md. Masudul Islam
Department of Computer Science & Engineering, Bangladesh University of Business & Technology, Bangladesh
E-mail: masudulislam11@gmail.com

Received: 12 August, 2022; Revised: 30 October, 2022; Accepted: 15 November, 2022; Published: 08 February, 2023

Abstract: In a combinatorial auction, there has a server, some agents, and some jobs which can be used to reach efficient resource and job allocations among the agents. In our paper, we have shown how any server can achieve maximum throughput as well as maximum profit based on some server constraints where each agent has one or more skills to perform those jobs on a priority basis which can be executed in a whole or partial. This algorithm can effectively distribute the appropriate job allocation among skilled agents with proper acknowledgment to the server after a certain period.

Index Terms: Agents, job allocation, server, scheduling, optimal, algorithm, combinatorial auction.

1. Introduction

Nowadays the parallel processing and distribution systems are very common issues in computer science. Scheduling has become a major factor for parallel and distributive systems to get better performance. This introduces some challenging problems in recent years and the improvement of scheduling algorithms for different scenarios is real [1]. There are two main estimations of scheduling algorithms—*scheduling performance* and *scheduling efficiency*. Scheduling performance is a minimum total completion time of a program. Efficiency is the measurement of the algorithm—how much capability the algorithm must acquire the goal [2]. These evaluations contradict each other. Job scheduling is a common problem in many different applications. However, no algorithm can solve this problem exactly finding the optimal solution in a reasonable time for any large configuration of jobs and agents. Regarding this challenge, we have devised possibly the most efficient algorithm to solve the problems of job allocation for the workers under a server constraint.

We have found a near similar work entitled “Optimization Techniques for Task Allocation and Scheduling in Distributed Multi-Agent Operations” when studying job scheduling. But the major differences between our research work and the work mentioned above [5] are in server constraints and agent skills are discussed later. This paper covers several advantages: maximizing profit, maximizing throughput, data synchronization for job completion reports, etc. that can be appended with multilevel queue scheduling [3]. Here, the term *data synchronization* refers to the server or processor a job status report that whether an agent or worker completed its task or not after a certain time. This *data synchronize* time may be considered as break time of an agent for sometimes in some applications.

2. Background

Before analyzing our proposed algorithm let’s take a tour of some commonly used scheduling models that almost and sometimes fully go with our research work but are not fully applicable [7]. The Multilevel Feedback Queue supports a job that can move among the various queues and scheduling policy for each queue. But it does not go with the queue migration mechanism. Moreover, Generalized Assignment problems refer to determining assignments of job quantities to agents with objectives are to minimize the total cost of the assignment and to reduce the time taken to complete all the jobs.

2.1. Multilevel Feedback Queue

In computer science, a multilevel feedback queue [8] is a scheduling algorithm. In the multilevel feedback queue, a process is given just one chance to complete at a given queue level before it is forced down to a lower-level queue. It is intended to meet the following design requirements for multimode systems and maximizing throughput:

1. Give preference to short jobs.
2. Give preference to I/O bound processes.
3. Separate processes into categories based on their need for the processor.

Multiple FIFO queues are used, and the operation is as follows:

1. A new process is positioned at the end of the top-level FIFO queue.
2. At some stage, the process reaches the head of the queue and is assigned the CPU.
3. If the process is completed it leaves the system.
4. If the process voluntarily relinquishes control, it leaves the queuing network, and when the process becomes ready again it enters the system on the same queue level.
5. If the process uses all the quantum time, it is pre-empted and positioned at the end of the next lower-level queue.
6. This will continue until the process completes or it reaches the base level queue.
7. At the base level queue, the processes circulate in a round-robin fashion until they are complete and leave the system.
8. Optionally, if a process blocks for I/O, it is 'promoted' one level, and placed at the end of the next-higher queue. This allows I/O bound processes to be favored by the scheduler and allows processes to 'escape' the base level queue.

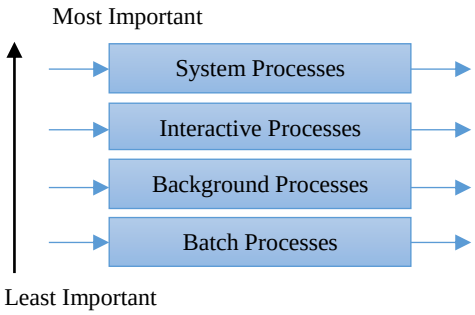

Fig. 1. Multilevel Feedback Queue.

In this system see Fig. 1, bursts are automatically placed in a lower priority queue if they fail to complete execution on their first attempt. Our proposed method allocates the various quantity of a task in several queues of agents by pre-calculating the status, capacity, processing power i.e., the skill of agents. Each agent takes the task from its queue and no tasks are pre-empted. Because the number of quantities of a job assigned to those queues tactically that follow maximum profit and throughput like a greedy approximation.

2.2. Generalized Assignment Problem

In applied mathematics, the maximum generalized assignment problem [9] is a problem in combinatorial optimization. This problem is a generalization of the assignment problem in which both jobs and agents have a size. Moreover, the size of each job might vary from one agent to the other. According to this problem, there are several agents and several jobs. Any agent can be assigned to perform any job, incurring some cost and profit that may vary depending on the agent-job assignment. Moreover, each agent has a budget and the sum of the costs of jobs assigned to it cannot exceed this budget. It is required to find an assignment in which all agents do not exceed their budget and the total profit of the assignment is maximized.

In the following, we have n kinds of items, x_1 through x_n and m kinds of bins b_1 through b_m. Each bin b_i is associated with a budget w_i. For a bin b_i, each item x_j has a profit p_{ij} and a weight w_{ij}. A solution is a subset of items U and an assignment from U to the bins. A feasible solution is a solution in which for each bin b_i the total weight of assigned items is at most w_i. The solution's profit is the sum of profits for each item-bin assignment. The goal is to find a maximum profit feasible solution. Mathematically the generalized assignment problem can be formulated as:

$$\text{Maximize} = \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} x_{ij} \tag{1}$$

$$\text{Subject To} = \sum_{j=1}^{n} w_{ij} \quad x_{ij} \leq w_i \quad (i = 1, 2, \dots, m) \tag{2}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1 \quad (j = 1, 2, \dots, n) \tag{3}$$

$$x_{ij} \in \{0, 1\} \quad (i = 1, 2, \dots, m) \,\&\, (j = 1, 2, \dots, n) \tag{4}$$

This generalized assignment problem can be formulated for our work as the goal is to find out partitions or subsets of jobs and an assignment from subset to agents (Fig. 2).

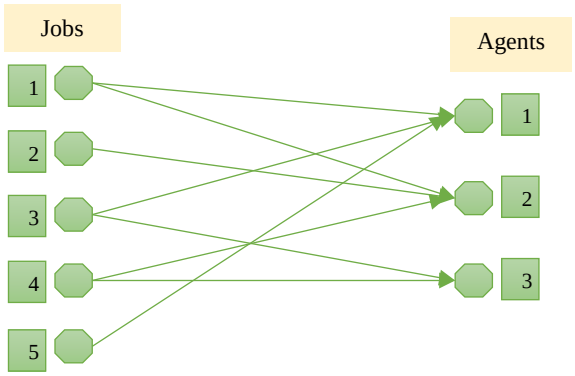

Fig. 2. Generalized Assignment Problem

Now if we map bins with agents and items with jobs then the formula mentioned above may be rewritten for maximizing profit as follows

$$\sum \boldsymbol{V_i * Q_i} \tag{5}$$

Where Q is the quantity for a job and V is the profit value for each quantity.

2.3. Greedy Approximation Algorithm

Using any algorithm ALG α-approximation algorithm for the knapsack problem, it is possible to construct an $(\alpha+1)$-approximation for the generalized assignment problem in a greedy manner using a residual profit concept [6]. The algorithm constructs a schedule in iterations, where during iteration J_i tentative selection of jobs to agent A_i is selected. The selection for agent A_i might change as jobs might be reselected in a later iteration for other jobs. The residual profit of a job J_i for agent A_i is P_{ij} if J_i is not selected for any other agent or (P_{ij} - P_{ik}) if J_i is selected for agent A_k. Formally we can state that,

Set: A_i = -1 for all i= 1…., n and j=1…., m
Do: Call ALG to find a solution to agent A_j using the residual profit function P_j.
Denote the selected jobs by S_j.
Update A using S_j, where $A_i = j$ for all $i \in S_j$

3. Problem Statement

In our problem scenario, we have set some criteria for our constraints and dataset. According to our scenario, agents can do any percent of a job if it does not precisely fit within a time frame which is the allocated time between data synchronization with the server or it may be the time between the agent's idle times. We have considered that agents normally have so many different skills, and the total number of skills can range between 1-100. We take the number of agents is 1to 1000 and the number of jobs is up to a million.

Let us assume that there are Jobs:

$$(J_1, Q_1, V_1), (J_2, Q_2, V_2), \dots \dots \dots (J_n, Q_n, V_n)$$

Where J_i is the job type, Q_i is the quantity, V_i is the value of the job for each quantity. Also, there are Agents with skills

Such as $\boldsymbol{A_1}$: $(J_1, T_1, S_1, St_1, P_1), (J_2, T_2, S_1, St_1, P_2)$ and $\boldsymbol{A_2}$: $(J_2, T_2, S_2, St_2, P_1), (J_3, T_3, S_2, St_2, P_2)$

Where J_i is the job the agent can do, T_i is the time of the agent doing each quantity of J_i, S_i is the time frame within which the agent can perform data synchronization exactly once with the server, St_i is the time the agent consumes during synchronization each quantity of J_i, and P_i is the job priority to the server. From the statement, we see that the

job table can be ordered on the value both for ascending and descending manners. Hence, sometimes job table can be considered for cost factor but for now, this is arranged for maximum profit value. On the other hand, an agent can handle more jobs on a priority basis and consumes some time to perform each quantity of a job. Moreover, two or more agents can also perform the same job while they are consuming separate time, and they have a different priority for the job.

3.1. Server Constraints

We have considered the following constraints for the server when developing the algorithm:

1. This scheduler algorithm must follow the hierarchy of jobs by maximizing profit value.
2. This algorithm should maximize throughput when allocating jobs to the agents.
3. A job can be partitioned in different quantities for a specific agent.
4. Each agent must acknowledge with the server after a specific time that tells what it has completed or not.
5. A job will be preempted if an agent remains busy or inactive.

3.2. Analysis and Calculations

Suppose a job table contains the following entries ordered by value in descending i.e., maximum to minimum profit manner (Table 1)

Table 1. Job Table

Job (J)	Quantity (Q)	Value (V)
1	8	10
2	3	8
3	4	3

Agents are containing the following entries in Table 2

Table 2. Agent's schedule and Priority

A_1				
Job (J)	Job Completion Time (T)	Syn. Interval (S)	Syn. Time (S_t)	Priority (P)
1	5	25	1	1
2	6	25	1	2
		A_2		
2	8	25	1	1
3	6	25	1	2
		A_3		
1	8	25	1	1
3	7	25	1	2
2	7	25	1	3

From the data table above we see that job J_1 was executed by agent A_1, A_3. Here both agents are capable to do the job in the same priority level P, but the difference is in execution time T. Agent A_1 consumes less time than agent A_3. Again, job J_2 was executed by three agents A_1, A_2, and A_3. Here the jobs are placed in different priority levels P and different execution times for three agents. Job J_3 is executed by agents A_2 and A_3 where the priority value is the same but execution time. If we analyze the total scenario then we can think of this as the following queuing systems.

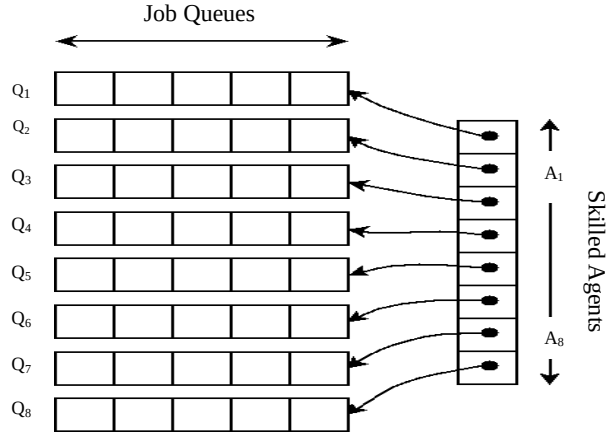

Fig. 3. Job queue System

Now our work is to manage the queues of each agent from job tables under server constraints. *A* queue can be filled up by the total quantity of a job or by partial quantity. To estimate the quantity of a job an agent can do within synchronization time interval we have to satisfy the following condition:

$$\text{Total Time, } (tT_{ij}) = T_i + St_j \tag{6}$$

Where **i**th job which is distributed to the **j**th agent.

$$\text{Selected Job Quantity, } (JQ_{ij}) = \lceil (Q_j/f) \rceil \tag{7}$$

Where **Q** is total quantity and **f** denotes the number of a job partitioned in quantities
So, the following condition must satisfy when partitioning job quantity for an agent.

$$(Q_{ij} \times tT_{ij}) \leq S_j \tag{8}$$

Where **S** is the synchronization time.
Each agent reads its private queue and executes the assigned job from the front of the queue. Hence, to support data synchronization by the agent we assign a value *synval=0* tactically in the queue. So, when an agent fetches the *synval* from its queue it goes to acknowledgment state with the server. For this circumstance, we must satisfy the following condition,

$$(synval_{ij} \times tT_{ij}) \geq S_j \tag{9}$$

Where ***synval*** associated with enqueue operation and it increases from 0 to $\boldsymbol{Q_{ij}}$ within the synchronization time frame for each entry in the queue.

4. Proposed Scheduling Algorithm

At the beginning of algorithm execution, all jobs were grouped into records (Table I). Each group contains job numbers and quantities. Jobs are sorted by maximum to minimum profit value. Each agent table (Table II) also contains a list of jobs that represents the skills of the agent. To get the best result, we have applied the following steps:

1. At first, we have found out the total number of jobs and the total number of agents under the server.
2. Then we have made a list of agents containing the attributes of execution time (T), synchronized interval time (S), synchronization time (S_t), and priority (P) for each job (J_i).
3. The list is sorted on a priority basis (from high to low) and then according to less execution time for the job.
4. At last, the job quantities are distributed to the agent's private queue by a scheduling sub-algorithm *make_schedule*.

We have partitioned the complete algorithmic steps into two parts—the main algorithm and the sub-algorithm. The main algorithm called ***scheduler*** is given below.

jt denotes sorted job table,
at denotes agent table,
agent_len refers to the number of agents,
job_len refers to the number of jobs,
aq refers to agent queues,
active refers agent status,
syn refers to counting data synchronization interval,
cjob refers to currently selected job,
Qty indicates selected job quantity,
K is the number of agents skilled for a specific job,
aglist indicates selected agents for a job,
job is filtered job list referencing by agents.
Step 1: calculate ***agent_len*** and ***job_len***, the total number of records from ***at*** and ***jt*** respectively.
Step 2: Repeat step 3 for c=1 to ***agent_len*** for each agent's queue ***aq***.
Step 3: set $front_c$=-1, $rear_c$=-1, $full_c$=0, $active_c$=1, and syn_c=0.
Step 4: Repeat step 5, step 6, step 9, and step 10 for i=0 to ***job_len*** for each job in ***jt***.
Step 5: set $\boldsymbol{cjob_i}$ of job= J of $\boldsymbol{jt_i}$, Qty_i=Q of $\boldsymbol{jt_i}$, and k=0.
Step 6: Repeat step 7 for j=1 to ***agent_len*** for each agent in ***at***.
Step 7: if J of $\boldsymbol{at_j}$ = J of $\boldsymbol{jt_i}$ go to step 8.

Step 8: add $aglist_i$ of job= A of at_j; set k=k+1.
Step 9: sort $aglist_i$ first by priority P and then by execution time T.
Step 10: call sub-procedure ***make_schedule****(Qty$_i$ of job, 1, 1, k, i)*.
Step 11: Repeat step 12 for i=1 to ***agent_len*** for each agent's queue ***aq***.
Step 12: Queue out the front value from aq_i and print it which indicates a job J.
Step 13: Exit

The following sub-algorithm assigns a job to different agents by maintaining server constraints recursively. It applies the following steps.

1. At first, it calculates the quantity of a job to check whether it can be held within the synchronization time frame or not.
2. This calculated job quantity goes to check all the agents whether it is fit to assign in a time frame or not.
3. These two steps mentioned above repeats recursively by partitioning job quantity for each iteration until it assigns a job to agents successfully.

A full sub-algorithm called ***make_schedule*** is given below.

Q denotes the quantity of a job **J**,
f denotes the number of a job partitioned in quantities,
A refers an agent,
$\mathbf{A_n}$ refers the number of agents skilled for a job **J**,
aq refers to agent queues,
active refers agent status,
syn for counting data synchronization interval,
cjob refers to currently selected job,
Qty indicates selected job quantity,
aglist indicates selected agents for a job,
job=filtered job list referencing by agents,
JQ refers to job quantity for each partition,
tT= total execution time and synchronous time for a job, **S**=data synchronization interval time.
Step 1: if A<=A_n then execute step 2 to step 9; else go to step 10.
Step 2: set **JQ**=ceil(***Q/f***), $tT=T_{AJ}+St_{AJ}$.
Step 3: if $JQ*tT<=S_{AJ}$ then go to step 4; else go to step 9.
Step 4: Repeat step 5 for i=0 to ***JQ*** for each quantity in ***J***.
Step 5: if aq_A is not full or ***A*** is active then execute step 6 and step 7 else break.
Step 6: insert $cjob_j$ of the job into aq_A and set ***Q=Q-1***.
Step 7: if $syn_A*tT>=S_{AJ}$ then set front$_A$ of ***aq***=0 and insert zero (0) into aq_A.
Step 8: if ***Q***>0 then call ***make_schedule****(Q,1, ++A, A_n, J);* else return.
Step 9: if ***Q***>0 then call ***make_schedule****(Q, f,++A, A_n, J)*; else return.
Step 10: if ***Q***>0 then call ***make_schedule****(Q, ++f,1, A_n, J);* else return.
Step 11: Exit

5. Results

After implementing the designed algorithm in programming language C, the following Fig. 4 shows the sorted list of skilled agents for each job for the data set given in the problem definition. The figure illustrates that there are three jobs with some quantities. Each job provides a list of agents who are skilled for that job sorted by priority and time.

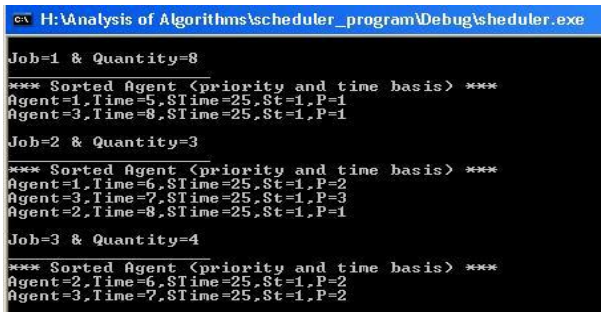

Fig. 4. Sorted list of skilled agents for each job

The following Fig. 5 shows the job quantity distributed among the agents. In this figure, the consecutive values (i.e., 1,1,1,1,1,0,1, 2, etc.) for agent A_1 queue refers that, at the first agent, A_1 executes five quantities of job J_1. Then A_1 acknowledge its completion status with the server by zero (0) values in the queue and continues by this manner for rest of the time. The same procedure is applicable for the other agents A_2 and A_3.

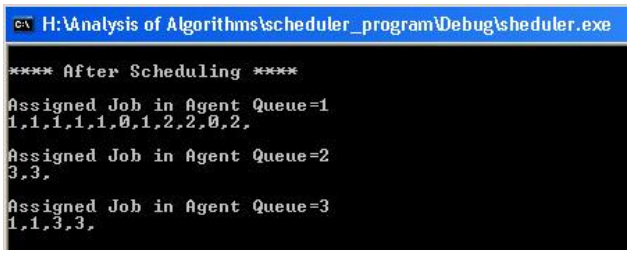

Fig. 5. Job quantity distributed among the agents

Now if we compare the above two figures with the given data set, we see that agents are precisely sorted one after another for each job Fig. 4. Again, total quantities are correctly distributed to the skilled agents by maintaining proper server constraints. The following Fig. 6 clearly illustrates the distribution of jobs according to time.

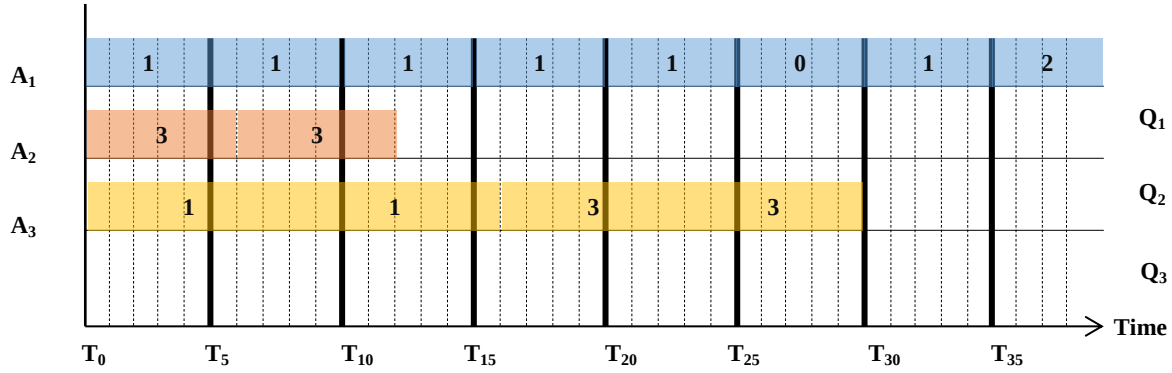

Fig. 6. Distribution of jobs concerning the time

6. Conclusion

In this article, a new approach has been proposed to allocate jobs for a skilled agent under a server constraint. Although this work shows a good solution for static scheduling, it also faces complexity. We say this is the limitation of this work as this algorithm checks each agent iteratively for a partitioned job quantity to fit and so it splits the total quantity for other agents while they may not exist. But this algorithm has real implementation where there are some workers/ agents, and they must acknowledge their activities after a certain period of what they have done to the server.

In the future, we can implement this approach to the most common system—automatic SIM toolkit (STK) management system for multiple mobile operators because a mobile agent can perform various activities against a SIM number. A SIM number can act for sending bulk SMS, or sometimes it can act for paying the utility bill, etc. Another implementation can be the automatic slot assignment for any task such as university routine generate for faculties.

References

[1] Klaus Jansen, Lars Prädel, Ulrich M. Schwarz, Ola Svensson, “Faster Approximation Algorithms for Scheduling with Fixed Jobs,” Proceedings of the Seventeenth Computing on The Australasian Theory Symposium, vol. 119, pp. 3–10, 2011.

[2] “Scheduling Algorithms,” Springer.com. [Online]. Available: https://www.springer.com/gp/book/9783540695158. [Accessed: 19-Aug-2021].

[3] L. Kleinrock and R. R. Muntz, “Processor sharing queueing models of mixed scheduling disciplines for a time-shared system,” J. ACM, vol. 19, no. 3, pp. 464–482, 1972.

[4] O. Rusanova and A. Korochkin, “Scheduling problems for parallel and distributed systems,” ACM SIGAda Ada Lett., vol. XIX, no. 3, pp. 195–201, 1999.

[5] Tompkins, "Optimization techniques for task allocation and scheduling in distributed multi-agent operations", Hdl.handle.net, 2021. [Online]. Available: http://hdl.handle.net/1721.1/16974. [Accessed: 21- Aug- 2021].

[6] J. Bar-Ilan, G. Kortsarz and D. Peleg, Greedy approximation algorithms. Rehovot, Israel: Weizmann Institute of Science, Dept. of Applied Mathematics and Computer Science, 1992.

[7] “Scheduling algorithms,” Ctl.ua.edu, 2012. [Online]. Available: http://www.ctl.ua.edu/math103/scheduling/scheduling_algorithms.htm. [Accessed: 19-Aug-2021].

[8] Wikipedia contributors, "Multilevel feedback queue," Wikipedia, The Free Encyclopedia, 11-Jan-2021. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Multilevel_feedback_queue&oldid=999660595. [Accessed: 19-Aug-2021].
[9] Wikipedia contributors, "Generalized assignment problem," Wikipedia, The Free Encyclopedia, 27-Nov-2020. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Generalized_assignment_problem&oldid=990955957. [Accessed: 19-Aug-2021].

Authors' Profiles

Mijanur Rahaman, Assistant Professor in Dept. of CSE in Bangladesh University of Business & Technology. His main area of working is networking, cryptographic security system and software base automation system. He is the developer and controller "Student Information and Management System" software of current university. He was the main webmaster and master analyzer of ACM-ICPC regional Dhaka site 2014 also he was chair of technical committee of ICPC Regional Dhaka Site 2021.

Md. Masudul Islam, a 33 years old Teacher, Researcher & former Web Developer from Bangladesh. He has been working as a teacher at Bangladesh University of Business & Technology (BUBT) in department of CSE for 9 years. His current position is Assistant Professor, Dept of CSE. He is doing Ph.D in department of CSE, Jahangirnagar University. He loves everything that has to do with teaching, IT related works, Data Science, Web Programming, Astronomy, Database, Quantum Computing, History, Religion and System Analysis. He feels a true devotion for teaching and research. His passion is to be a good teacher as well as researcher.

How to cite this paper: Mijanur Rahaman, Md. Masudul Islam, "Optimal and Appropriate Job Allocation Algorithm for Skilled Agents under a Server Constraint", International Journal of Education and Management Engineering (IJEME), Vol.13, No.1, pp. 10-17, 2023. DOI:10.5815/ijeme.2023.01.02