Hindawi
Mathematical Problems in Engineering
Volume 2021, Article ID 5803092, 8 pages
https://doi.org/10.1155/2021/5803092

Research Article

A Network Flow Algorithm for Solving Generalized Assignment Problem

Yongwen Hu [1,2] **and Qunpo Liu** [3,4]

[1]*Key Laboratory of Power System Design and Test for Electrical Vehicle, Hubei University of Arts and Science, Xiangyang 441053, China*
[2]*School of Mechanical Engineering, Hubei University of Arts and Science, Xiangyang 441053, China*
[3]*Department of Robotics Engineering, Henan Polytechnic University, Jiaozuo 454003, China*
[4]*Henan International Joint Laboratory of Direct Drive and Control of Intelligent Equipment, Jiaozuo 454003, China*

Correspondence should be addressed to Qunpo Liu; lqpny@hpu.edu.cn

Received 14 July 2020; Revised 24 August 2020; Accepted 24 December 2020; Published 12 January 2021

Academic Editor: M Javaid

The generalized assignment problem (GAP) is an open problem in which an integer k is given and one wants to assign k' agents to $k\,(k' \le k)$ jobs such that the sum of the corresponding cost is minimal. Unlike the traditional $\mathscr{K}$-cardinality assignment problem, a job can be assigned to many, but different, agents and an agent may undertake several, but different, jobs in our problem. A network model with a special structure of GAP is given and an algorithm for GAP is proposed. Meanwhile, some important properties of the GAP are given. Numerical experiments are implemented, and the results indicate that the proposed algorithm can globally and efficiently optimize the GAP with a large range cost.

1. Introduction

Assignment problem (AP) is one of the fundamental combinatorial optimization problems with various applications in real life. The classical AP is proved to be an $\mathscr{NP}$-hard problem, and it deals with a situation of assigning n jobs to n agents such that each job must be processed by exactly one agent and vice versa. And the popular systematic method for solving classical AP is Kuhn's Hungarian method [1, 2] which can be extended for general network flow problems in a polynomial time. However, it is quite difficult to ensure that the number of jobs is exactly equal to the number of agents in real-life situation. Furthermore, jobs and agents are set to be unique in the classical AP, which leads to any row (column) in $c_{ij}, \forall i \in \mathbf{M}, j \in \mathbf{N}$ ($\mathbf{M}$ is an index set of agents and $\mathbf{N}$ is an index set of jobs) being different from every other row (column). In fact, although all agents are unique, some jobs can be identical and vice versa in practical applications. Kennington and Wang [3] give some typical problems, such as manpower planning, scheduling, and planning, in which more than one agent is assigned to the same job group. Also, a shortest augmenting path algorithm is presented for solving them.

Dell'Amico and Martello [4] considered a generalization of AP (called $\mathscr{K}$-cardinality linear assignment problem) where one wants to assign k (out of m) agents to k (out of n) jobs $(k \le \min(m, n))$ so that the sum of the corresponding cost is minimal. Some potential applications of $\mathscr{K}$-cardinality linear assignment problem can be found in [5, 6]. Further, the $\mathscr{K}$-cardinality assignment problem has various applications, for example, in assigning agents to machines when there are multiple alternatives and only a subset of agents and machines has to be assigned. In this paper, we consider a more generalized assignment problem (GAP) in which a cost matrix $A_{m\times n}$ and positive integer $k\,(0 < k \le \min\{m, n\})$ are given and one wants to assign k' agents to $k\,(k' \le k)$ jobs so that the sum of the corresponding cost is minimal.

Dell'Amico and Martello developed some algorithms for solving $\mathscr{K}$-cardinality linear assignment problem by min-

cost flow or shortest augmenting path techniques described in [4]. Also, heuristic processing techniques were given in [7]. Later, Volgenant [8] developed an algorithm for it by solving a sequence of classical AP. Recently, based on semi-Lagrangian relaxation, Belik and Jö rnsten [9] proposed an algorithm for the $\mathcal{K}$-cardinality AP with large scale. Clearly, the GAP considered in this paper is the $\mathcal{K}$-cardinality assignment problem when $k' = k$. The GAP is an $\mathcal{NP}$-hard [10] problem. Later, Chu and Beasley [11] prove that GAP is $\mathcal{NP}$-complete.

Several algorithms have been proposed for GAP in the past few years. Most of the small sizes of the test instances have been solved by some exact methods such as branch and bound, branch and price, and cutting plane [12–16]. However, the exact methods are mostly unable to find an optimal solution within a reasonable computational time. Metaheuristic and heuristics have been proposed for solving GAP in order to reduce the computational time of the exact method, such as the combination of a heuristics and exact method [17], tabu search [18], simulated annealing [19], genetic algorithm [11, 20], differential evolution algorithm [21, 22], and bees algorithm [23].

Clearly, the GAP considered in this paper can be transformed into a classical AP when some dummy jobs or agents are introduced. And the Hungarian method and its variants [24–26] are presented for optimizing GAP recently. However, the transformation will result in a larger-scale problem and worse space complexity. Currently, several exact algorithms are presented for solving GAP with minimum cost, such as variable-fixing algorithm [27], branch and bound algorithm [12], and KM algorithm [28], but these algorithms can only solve GAP with some specific assumption.

This study focuses on GAP where the number of agents is not equal to the number of jobs. Specifically, one job can be assigned to many, but different, agents, and one agent may undertake many, but different, tasks. It is well known that AP is thought of as a special min-cost flow problem, which suggests that classical AP or GAP can be solved by some methods with network flow theory. Recently, an efficient algorithm was proposed for solving min-cost flow problem [29]. This paper attempts to give a variation of the algorithm presented in [29] for GAP based on the advantage of the special structure of the transformed network model.

This paper is organized as follows. Section 2 gives a mathematical formulation and network flow model for the GAP; in Section 3, we describe an algorithm for solving GAP. Also, the convergence of the proposed algorithm and some important features of GAP are given in this section; Section 4 presents numerical experiments to evaluate the performance of the proposed algorithm. Concluding remarks and some further research are provided in Section 5.

2. Mathematical Formulation and Network Flow Model

Given an $m \times n$ matrix $C = [c_{ij}]$ and an integer $k\,(0 < k \le \min\{m, n\})$, the GAP can be formulated as follows:

$$\text{GAP}\begin{cases} \min \quad z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \\ \qquad \sum_{j=1}^{n} x_{ij} = y_i, \quad i = 1, 2, \ldots, m, \\ \qquad \sum_{i=1}^{m} x_{ij} = z_j, \quad j = 1, 2, \ldots, n, \\ s.t. \quad \sum_{i=1}^{m} y_i = k, \\ \qquad \sum_{j=1}^{n} z_j = k, \\ \qquad x_{ij} = 0 \text{ or } 1, \quad i = 1, 2, \ldots, m;\ j = 1, 2, \ldots, n, \end{cases} \tag{1}$$

where $k, y_i, z_j \in I; c_{ij} \ge 0; i \in \mathbf{M} = \{1, 2, \ldots, m\}; j \in \mathbf{N} = \{1, 2, \ldots, n\}$; $x_{ij} = 1$ if agent i is assigned to job j. First, we consider the case that an agent can be assigned to more than one job, but a job is assigned to exactly one agent, that is, $z_j = 1$, $\forall j \in \mathbf{N}$ in (1). We leave the situation that a group job can be processed by more than one agent later.

It is well known that classical AP is a bipartite matching problem which can be transformed into a maximum flow problem in a simple network. To transform the GAP defined on an undirected graph $G = (\mathbf{M} \cup \mathbf{N}, \mathbf{A})$, we create a directed version of G by designating each arc in A as pointing nodes in $\mathbf{M}$ to nodes in $\mathbf{N}$. Then, we introduce a source node s and a sink node t, with $(s, i), \forall i \in \mathbf{M}$, and $(j, t), \forall j \in \mathbf{N}$. We refer to the transformed network as $G' = (\mathbf{N}', \mathbf{A}', \mathbf{C}, \mathbf{U})$ with a cost set $\mathbf{C}$ and a capacity set U associated with each arc $(i, j) \in \mathbf{A}'$. Let $c(s, i) = 0, c(j, t) = 0, 0 \le u(s, i) \le y_i$, $0 \le u(j, t) \le 1, 0 \le u(i, j) \le 1, \forall i \in \mathbf{M}, j \in \mathbf{N}$.

Figure 1 illustrates the transformation. Mathematically, the transformed model is given by

$$\text{MCF}\begin{cases} \min \quad z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \\ \qquad \sum_{i=1}^{m} x_{si} = k, \quad i \in \mathbf{M}, \\ s.t. \quad \sum_{j=1}^{n} x_{jt} = -k, \quad j \in \mathbf{N}, \\ \qquad \sum_{j} x_{ij} - \sum_{i} x_{ij} = 0, \ \forall i, j \in \mathbf{M} \cup \mathbf{N}, \\ \qquad 0 \le x_{ij} \le 1, \ \forall i, j \in \mathbf{M} \cup \mathbf{N} \cup \{s\} \cup \{t\}. \end{cases} \tag{2}$$

It is easy to deal with the case that some identical jobs can be grouped into a set. More specifically, if $1'$ is a group set which consists of $z_{1'}$ jobs, then the capacity of $(1', t)$ is to be set to $z_{1'}$ instead of 1.

According to the transformation mentioned above and the *Integrality Theorem* [30], it is easy to get the following theorem.

Theorem 1. *The GAP is equivalent to the MCF.*

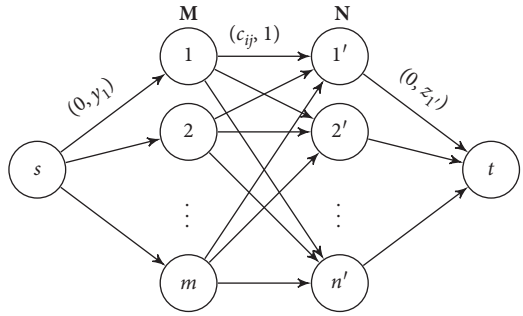

Figure 1: Unit capacity network flow model.

Theorem 1 indicates that we can find an integral flow of value k in the transformed network in order to get an optimal solution of GAP. Although several algorithms are presented for optimizing minimum-cost flow problem, both cost and capacity are to be considered for iteratively defining residual network. Hu et al. [29] introduced an approach that can solve min-cost flow. Unlike the traditional algorithms, the proposed algorithm in [29] can find an augmenting path in the original network by updating node potentials. We will describe an algorithm for optimizing GAP based on the network model defined above in the next section.

3. Proposed Algorithm for Solving GAP

Hu et al. [29] gave a detail algorithm for solving minimum-cost flow problem by introducing some optimality conditions. Since the transformed network flow model, shown in Figure 1, is a simple network, a modified version of the algorithm in [29] is introduced for GAP for self-contained in this section.

Let us consider a dual problem of problem (2)

$$\text{DP}\left|\begin{array}{ll} \max & w = kp_s - kp_t + \sum_{(i,j)\in \mathbf{A}'} u_{ij}p_{ij}, \\ & p_i - p_j + p_{ij} \le c_{ij},\ \forall (i,j) \in \mathbf{A}', \\ s.t. & p_i \text{ is free},\ \forall i \in \mathbf{M} \cup \mathbf{N} \cup \{s\} \cup \{t\}, \\ & p_{ij} \le 0,\ \forall (i,j) \in \mathbf{A}', \end{array}\right. \tag{3}$$

where p_i and p_{ij} are the dual variables of MCF. And we refer to p_i and p_{ij} as the potential of node i and arc $(i, j) \in \mathbf{A}'$, respectively.

Let x_{ij} and $p = \{p_i, p_{ij}\}$ be the feasible solutions of MCF and DP, respectively. If x_{ij} and $p = \{p_i, p_{ij}\}$ are optimal solutions, according to strong duality, we have

$$\begin{aligned} &x_{ij} = 0, \quad \text{if } p_i - p_j + p_{ij} < c_{ij},\ \forall i \in \mathbf{N}', \\ &x_{ij} = u_{ij}, \quad \text{if } p_i - p_j > c_{ij},\ \forall i \in \mathbf{N}',\ (i,j) \in \mathbf{A}'. \end{aligned} \tag{4}$$

Since p_i is free $\forall i \in \mathbf{N}'$, let $p_{ij} = \min\{0, c_{ij} + p_j - p_i\}$, then $p_{ij} \le 0$ will be held. Therefore, a feasible solution $p = \{p_i, p_{ij}\}$ of DP can be found. Thus, conditions (4) are equal to the conditions as follows:

$$\begin{aligned} &x_{ij} = 0, \quad \text{if } p_i - p_j < c_{ij},\ \forall i \in \mathbf{N}', \\ &x_{ij} = u_{ij}, \quad \text{if } p_i - p_j > c_{ij},\ \forall i \in \mathbf{N}',\ (i,j) \in \mathbf{A}'. \end{aligned} \tag{5}$$

Clearly, an optimal solution $x = x_{ij}$ will satisfy conditions (5). And we have the following theorem.

Theorem 2. *Let x_{ij} be a feasible solution of MCF, then x_{ij} is an optimal solution if and only if the following conditions are satisfied:*

$$\begin{cases} p_i - p_j < c_{ij} \Longrightarrow x_{ij} = 0, \\ p_i - p_j > c_{ij} \Longrightarrow x_{ij} = u_{ij}. \end{cases} \tag{6}$$

We refer the arc satisfying $p_i - p_j = c_{ij}$, $\forall (i, j) \in \mathbf{A}'$ to an admissible arc. Similarly, a network is an admissible network in which each arc is an admissible arc. Clearly, $x_{ij} = 0$, $\forall (i, j) \in \mathbf{A}'$ is an optimal solution with an integral flow of value 0. The proposed algorithm starts sending 0 unit of flow from s to t. Furthermore, by holding condition (6) at each iteration, the proposed algorithm can find at least an augmenting path from the source node s to the sink node t after updating node potential in finite iteration. Therefore, flow can be augmented when the current amount flow is not maximal.

Let μ^+ and μ^- be a forward arc set and a backward arc set on an augmenting set μ. Now we describe an algorithm for solving GAP corresponding the network $G' = (\mathbf{N}', \mathbf{A}', \mathbf{C}, \mathbf{U})$, where $\mathbf{N}' = \mathbf{M} \bigcup \mathbf{N}$, and we refer an admissible network to R.

Clearly, for an optimal solution $x^*_{ij}, \forall i \in \mathbf{M}, j \in \mathbf{N}$, agent $i (i \in \mathbf{M})$ will be assigned to job $j (j \in \mathbf{N})$ if $x^*_{ij} = 1$. We give the following definition in advance in order to get some important properties on GAP.

Definition 1. Let $\mathbf{V}_i$ and $\mathbf{V}_j$ be the node sets of any two augmenting paths (referred to μ_i, μ_j) from source node s to sink node t, then μ_i, μ_j are independent augmenting paths if $(\mathbf{V}_i \backslash \{s\} \backslash \{t\}) \cap (\mathbf{V}_j \backslash \{s\} \backslash \{t\}) = \varnothing$.

Theorem 3. *Suppose the current amount of flow obtained by Algorithm 1 is k' $(k' < k)$, GAP has $C_q^{k-k'}$ optimal solutions if we can find $q (q \ge k - k')$ independent augmenting paths in the following iteration.*

Theorem 3 can be proved easily according to the detail algorithm and definition of independent augmenting path mentioned above, and here, we do not discuss the proof of Theorem 3.

Theorem 4. *Let $c_{i',j'} = \min c_{ij}$, $\forall i \in \mathbf{M}, j \in \mathbf{N}$ be in a row set $\mathbf{R_o}$ and in a column set $\mathbf{C_o}$, then at least an agent $i' \in \mathbf{R_o}$ is to be assigned to a job in $\mathbf{N}$. Similarly, at least a job $j' \in \mathbf{C_o}$ is to be processed by an agent in $\mathbf{M}$.*

Initial settings: $p_i = 0$, $\forall i \in \mathbf{N}'$, $x_{ij} = 0$, $\forall (i,j) \in \mathbf{A}'$, $\mathbf{S} = \{s\} \bigcup \mathbf{M}$, $\overline{\mathbf{s}} = \mathbf{N}' \backslash \mathbf{S}$. Label source node $(0,1)$. Let k be a given integer number such that $0 < k \leq \min\{m,n\}$.
while $\sum_{i=1}^{m} x_{si} < k, \quad i \in \mathbf{M}$ **do**
  Remove all labels of node i, $\forall i \in \mathbf{M} \bigcup \mathbf{N} \bigcup \{t\}$.
  while $t \notin \mathbf{S}$ **do**
    $p_i = p_i + \theta$, $\forall i \in \mathbf{S}$, where $\theta = \min_{k \in \mathbf{S} \cap \mathbf{M}, q \in \overline{\mathbf{S}} \cap \mathbf{N}} c_{kq} - p_k + p_q$.
    if $p_i - p_j = c_{ij}$, $\forall i \in \mathbf{S}$, $j \in \overline{\mathbf{S}}$ **then**
      $(i,j) \in R$.
    end if
    if $(i,j) \in R$, $x_{ij} = 0$, $\forall i \in \mathbf{S}$, $j \in \overline{\mathbf{S}}$ **then**
      Node j is labeled with $(i,1)$, $\mathbf{S} = \mathbf{S} \bigcup \{j\}$, $\overline{\mathbf{S}} = \overline{\mathbf{S}} \backslash \{j\}$.
    end if
    if $(j,i) \in R$, $x_{ji} = 1$, $\forall i \in \mathbf{S}$, $j \in \overline{\mathbf{S}}$ **then**
      Node j is labeled with $(-i,1)$, $\mathbf{S} = \mathbf{S} \bigcup \{j\}$, $\overline{\mathbf{S}} = \overline{\mathbf{S}} \backslash \{j\}$.
    end if
  end while
  $$x_{ij} = \begin{cases} x_{ij} + 1, & (i,j) \in \mu^{+}, \\ x_{ij} - 1, & (i,j) \in \mu^{-}, \\ x_{ij}. \end{cases}$$
end while

Algorithm 1: Algorithm for GAP.

Proof. Since $\theta = \min_{k \in \mathbf{S} \cap \mathbf{M}, q \in \overline{\mathbf{S}} \cap \mathbf{N}} c_{kq} - p_k + p_q$, at least an admissible will be introduced after updating node potentials in the following iteration. Thus, $p_{i'} - p_{j'} = c_{i',j'}$, and $c_{i',j'} = \min c_{ij}$, $\forall i \in \mathbf{M}$, $j \in \mathbf{N}$, which implies that nodes s, i', j' can be labeled. In addition, as $c_{j't} = 0$, then sink node t is to be labeled. Therefore, we can find an augmenting path from s to t, and there exists at least a node $j' \in \mathbf{N}$ such that $x_{j't} = 1$. Furthermore, the equation $x_{j't} = 1$ is always satisfied in the following iterations according to the details in Algorithm 1. Thus, job j' is to be processed by an agent in $\mathbf{M}$ for optimal match. Similarly, for an optimal solution, there exists an agent $i' \in \mathbf{M}$ to be assigned to a job in $\mathbf{N}$. □

From Theorem 4, we have the following corollary.

Corollary 1. *Let x_{ij}^{*}, $i \in \mathbf{M}$, $j \in \mathbf{N}$ be an optimal solution for the GAP with $k-1$ cardinality, then the agents and jobs corresponding to $x_{ij}^{*} = 1$ will be assigned in GAP with k cardinality.*

Theorem 5. *Algorithm 1 will terminate in finite iterations.*

Proof. Clearly, sink node t cannot be labeled if $x_{jt} = 1$, $\forall j \in \mathbf{N}$. Otherwise, as $c_{si} = c_{jt} = 0$, $\forall i \in \mathbf{M}$, $j \in \mathbf{N}$, t will be labeled while j is labeled, which implies that we can find an augmenting path from s to t and one more unit flow can be sent from s to t at first iteration. In addition, node i can be labeled if j is labeled and $x_{ij} = 1$. Therefore, we can find an augmenting path after updating node potential in l iterations at l-th iteration, and at least one more unit flow can be sent from s to t. For GAP with k cardinality, the total number of iterations is $1 + 2 + \cdots + k = ((k^2 + k)/2)$, which gives Theorem 5. □

As it is described in Algorithm 1, flow can be augmented after updating nodes' potential in a finite iteration. Therefore, the complexity of the proposed algorithm is $O(k^2 (n+m)^2)$.

Next, we will give some numerical experiments to evaluate the efficiency of Algorithm 1.

4. Numerical Experiments

In this section, the performance of the proposed algorithm is demonstrated by numerical experiments and we compare its performance with a PP-Lapjv algorithm [31] and bees algorithm for GAP. The bees algorithm is proposed by Ozbakir et al. [23] for the complex integer optimization problem. The PP-Lapjv algorithm is an improved version of Lapjv algorithm [32, 33]. The GAP is transformed into a balanced AP when we use the PP-Lapjv algorithm for solving GAP. For the sake of simplicity, the PP-Lapjv algorithm is called A_1. The bees algorithm and our algorithm presented in the paper are called A_2 and A_3, respectively. The presented algorithm is coded in MATLAB^R R2016b on a laptop computer with an Intel®Corei5^{TM}i5-7300HQ CPU @2.50 GHz RAM 8 GB and the operating system is Windows 10 64 bit. As shown in Figure 1, the unit capacity network flow model of the GAP has a special structure that there is only forward (i,j), $\forall i \in M$, $j \in N$. Thus, it is easy to generate a random network which is connected.

A series of numerical experiments is implemented on test problems differing in input data values of c_{ij}, and the c_{ij} is drawn randomly from a uniform distribution on the intervals $[0, 10^2]$, $[0, 10^5]$. This kind of generation is often encountered to evaluate algorithms for solving AP [34]. The GAP with different scales ((m,n) varying from $(20,20)$ to $(200,400)$) is considered. And each class consists of instances with k valued $0.2v, 0.4v, 0.6v, 0.8v, 0.9v, v$, where $v = \min\{m,n\}$. The column titled k provides the coefficient of v. Also, for each class of problem, we consider the case that an agent can be assigned to k jobs when k is given. In addition, the proposed algorithm is evaluated on various

Table 1: Computational behavior of randomly generated GAP with various values of m, n, k, c, and a density valued 0.8 (times in seconds).

		$d\% = 80\%$						$d\% = 20\%$					
(m,n)	k	$c \in [0, 10^2]$			$c \in [0, 10^5]$			$c \in [0, 10^2]$			$c \in [0, 10^5]$		
		A_1	A_2	A_3	A_1	A_2	A_3	A_1	A_2	A_3	A_1	A_2	A_3
	0.2	5.620	0.046	0.040	5.940	0.047	0.014	4.971	0.040	0.012	5.837	0.043	0.006
	0.4	5.683	0.048	0.019	6.168	0.048	0.021	5.008	0.042	0.017	5.906	0.045	0.016
	0.6	5.926	0.052	0.052	6.186	0.053	0.035	5.255	0.045	0.021	6.183	0.050	0.018
(20, 20)	0.8	5.911	0.059	0.041	6.003	0.067	0.030	5.216	0.053	0.022	5.985	0.062	0.019
	0.9	6.098	0.060	0.050	6.064	0.072	0.040	5.343	0.053	0.029	6.090	0.066	0.024
	1.0	6.122	0.061	0.042	6.109	0.077	0.040	5.379	0.054	0.043	6.040	0.071	0.027
	Average	5.893	0.054	0.041	6.078	0.061	0.028	5.195	0.048	0.024	6.142	0.056	0.018
	0.2	5.855	0.108	0.040	6.187	0.111	0.025	5.168	0.095	0.011	6.007	0.108	0.015
	0.4	5.906	0.109	0.030	5.971	0.126	0.014	5.173	0.097	0.018	5.955	0.115	0.016
	0.6	6.101	0.112	0.070	6.128	0.128	0.032	5.382	0.099	0.023	5.910	0.119	0.020
(20, 40)	0.8	6.201	0.127	0.053	5.986	0.136	0.021	5.449	0.113	0.026	5.922	0.139	0.025
	0.9	6.440	0.135	0.063	6.030	0.139	0.045	5.705	0.123	0.030	5.917	0.145	0.025
	1.0	6.361	0.139	0.047	5.947	0.143	0.041	5.644	0.123	0.031	6.000	0.151	0.027
	Average	6.144	0.122	0.051	6.042	0.131	0.030	5.423	0.108	0.023	5.952	0.130	0.021
	0.2	5.951	0.551	0.178	7.246	0.667	0.083	5.247	0.567	0.050	6.977	0.850	0.045
	0.4	6.071	0.554	0.263	7.322	0.681	0.123	5.394	0.586	0.097	7.274	0.867	0.064
	0.6	6.114	0.565	0.331	7.385	0.692	0.152	5.418	0.590	0.125	7.230	0.885	0.119
(50, 50)	0.8	6.935	0.582	0.358	7.411	0.696	0.194	6.098	0.603	0.127	7.411	0.886	0.120
	0.9	7.311	0.584	0.358	7.272	0.701	0.206	6.417	0.618	0.210	7.232	0.952	0.128
	1.0	8.068	0.607	0.384	7.398	0.732	0.214	7.074	0.651	0.242	7.187	0.984	0.111
	Average	6.742	0.574	0.312	7.339	0.695	0.162	5.941	0.603	0.142	7.222	0.899	0.098
	0.2	6.737	0.846	0.344	8.624	1.105	0.064	5.896	0.746	0.080	8.305	1.082	0.042
	0.4	6.956	0.852	0.544	8.714	1.112	0.142	6.155	0.751	0.127	8.529	1.114	0.089
	0.6	6.872	0.920	0.686	8.893	1.124	0.194	6.026	0.817	0.169	8.669	1.207	0.113
(50, 100)	0.8	8.446	0.959	0.723	8.989	1.137	0.207	7.427	0.846	0.191	8.773	1.239	0.138
	0.9	9.252	1.051	0.803	9.057	1.146	0.214	8.153	0.925	0.234	8.667	1.304	0.203
	1.0	11.576	1.112	0.884	9.051	1.368	0.227	10.136	0.977	0.239	8.961	1.348	0.213
	Average	8.307	0.957	0.561	8.888	1.165	0.175	7.314	0.884	0.173	8.651	1.216	0.133
	0.2	6.760	2.264	0.782	8.940	3.038	0.168	5.944	1.988	0.322	8.604	2.843	0.183
	0.4	6.849	2.368	2.697	10.983	3.398	0.479	6.051	2.107	0.517	10.566	3.055	0.257
	0.6	6.994	2.413	3.322	11.534	3.440	0.583	6.150	2.148	0.642	11.053	3.136	0.382
(100, 100)	0.8	8.673	2.891	3.545	11.692	4.230	0.630	7.621	2.602	0.663	11.375	3.929	0.465
	0.9	9.286	3.107	3.714	11.943	4.661	0.854	8.181	2.796	0.866	11.744	4.018	0.587
	1.0	11.676	3.268	3.727	11.926	5.033	0.882	10.255	2.983	1.092	11.588	4.237	0.616
	Average	8.373	2.719	2.965	11.170	3.967	0.559	7.367	2.422	0.667	10.822	3.536	0.415
	0.2	9.555	15.103	5.256	11.826	18.724	0.343	8.411	11.700	0.769	11.597	16.087	0.212
	0.4	9.084	15.217	8.981	12.132	20.537	0.715	7.957	12.150	1.225	11.781	16.940	0.424
	0.6	9.357	16.467	11.173	12.238	23.267	0.977	8.231	13.530	1.470	11.881	19.305	0.599
(100, 200)	0.8	12.586	18.525	11.642	15.406	25.901	1.003	11.050	15.725	1.495	15.023	22.801	0.550
	0.9	12.845	19.637	11.963	15.714	27.588	1.099	11.283	17.052	1.758	15.302	25.578	0.744
	1.0	17.236	20.208	12.195	20.010	31.029	1.155	15.115	18.187	1.775	19.212	27.281	0.932
	Average	11.777	17.511	10.202	14.554	24.508	0.882	10.341	4.724	1.145	14.193	21.212	0.577
	0.2	9.670	22.612	34.092	12.301	28.250	1.826	8.498	18.316	3.673	12.087	24.907	0.847
	0.4	9.782	23.107	59.843	12.213	29.217	3.158	8.571	19.173	6.142	11.859	26.842	1.707
	0.6	9.825	24.750	70.225	12.693	34.423	4.208	8.676	21.285	7.608	12.318	30.225	2.446
(200, 200)	0.8	13.742	27.695	73.291	16.275	37.131	5.208	12.045	23.364	7.748	15.784	32.478	2.434
	0.9	14.158	28.733	74.686	16.692	42.157	5.658	12.420	24.952	9.039	16.286	33.167	4.569
	1.0	18.843	31.203	76.188	21.245	45.208	7.306	16.530	27.007	10.678	20.643	35.247	4.576
	Average	12.670	26.350	64.723	15.237	36.064	4.561	11.123	22.350	7.481	14.830	30.478	2.763
	0.2	9.529	108.531	127.211	12.686	145.807	3.993	8.375	93.961	12.338	12.139	133.425	1.628
	0.4	10.327	113.421	218.291	12.719	154.826	7.552	9.055	94.692	20.583	12.450	134.219	3.125
	0.6	10.135	120.280	266.023	12.518	164.904	9.913	8.941	102.241	24.847	12.293	143.627	4.389
(200, 400)	0.8	15.123	133.514	275.431	18.226	172.927	11.495	13.271	105.821	25.050	17.707	151.292	4.306
	0.9	16.541	139.770	285.422	18.824	180.619	11.870	14.486	122.327	27.784	18.184	156.443	5.852
	1.0	23.668	151.022	286.511	26.359	194.231	11.990	20.766	129.863	28.139	25.409	160.126	6.457
	Average	14.221	127.356	243.148	16.889	168.886	9.946	12.482	108.151	23.124	16.364	146.522	4.248

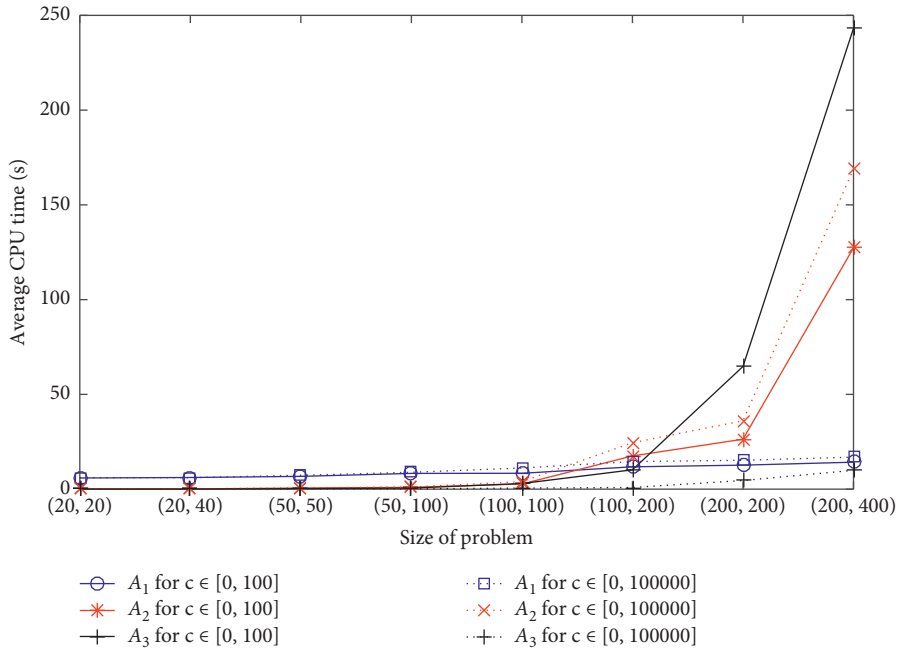

Figure 2: Average CPU time of algorithms A_1, A_2, and A_3 for different sizes of problem with density 80%.

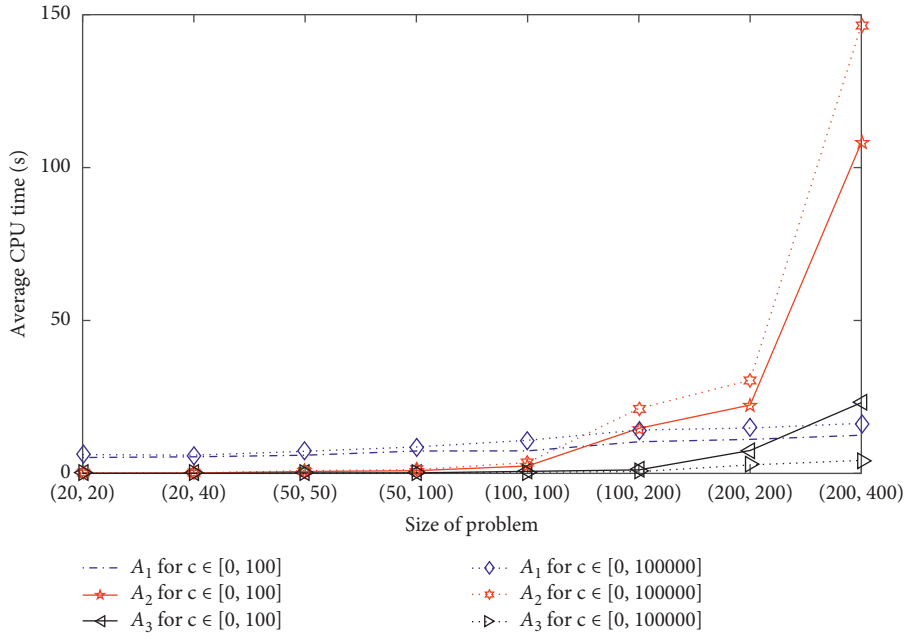

Figure 3: Average CPU time of algorithms A_1, A_2, and A_3 for different sizes of problem with density 20%.

degrees of density 20% and 80%. For a given density d%, the instance is generated as follows. We first randomly generate a value $r \in [0, 1]$, and for each arc $(i, j), \forall i \in M, j \in N$, the arc (i, j) is kept in the instance with a randomly generated cost c_{ij}; otherwise, the next arc is considered. Higher density means that each multiskill agent $i \in M$ can be assigned to more various jobs in M. Further, the parameter setting is the same as the parameters in [23].

We report the CPU time of A_1, A_2, and A_3 for solving GAP in Table 1. In each block, each entry in the 6 lines gives the average CPU time computed over 10 instances, neglecting the input, the output, and generation of the network flow model. And the last line gives the average over all k values considered. Also, Figures 2 and 3 depict the average CPU time on the various k values for the density 80% and 20%, respectively.

A_3 has a high probability to find an augmenting path from s to t after updating nodes' potential in two iterations, which can be easily verified by the network model and Algorithm 1 of GAP. Furthermore, as it is shown in

Algorithm 1, A_3 holds the information of node potential at each iteration. Therefore, for a larger range $c_{ij} \in [0, 10^5]$, the curves of the proposed algorithm are at the bottom of Figures 2 and 3. However, it will take more iterations for finding an augmenting path from s to t with a larger number of arcs $(i, j) \in A', i \in M, j \in N$ and a smaller range $c_{ij} \in [1, 10^2]$, $(i, j) \in A'$ for A_3. Thus, there are exceptions for the efficiency of A_3 when the size of GAP is increased (e.g., (m, n) is greater than (100, 100) for a smaller c_{ij}). It is understandable that A_3 will find an augmenting path in much more iteration in a network with a smaller c_{ij}, especially for a larger size of problem. In addition, larger size of GAP results in a greater number of local optimal solutions. Thus, it will take much more CPU time for A_2, which is shown in Figures 2 and 3.

The CPU time of algorithm A_1 consists of time for running Lapjv.m file and constructing the k-th transferred cost matrix which is mainly related to the size of the problem. Therefore, variation of CPU time of A_1 is much smaller than that of A_3 for the same size of problem (the size of the problem is defined by $\max\{m, n\}$).

5. Conclusion and Further Work

This paper develops an algorithm for solving GAP in which an integer k is given and one wants to assign k' $(k' \leq k)$ agents to k jobs such that the sum of the corresponding cost is minimal. The problem is transformed into a network model with a special structure. And the network model of GAP can easily process the GAP in which one agent can be assigned to more than one job or a job group can be processed by more than one agent. Some important properties of GAP are derived and numerical experiments show that our proposed algorithm has some good performance for solving GAP with a larger range cost c_{ij} and a smaller number of arcs (i, j), $\forall i \in M, j \in N$.

In this study, the scale of GAP is not very large, and we plan to develop some techniques for processing large-scale sparse cost matrix in order to optimize the GAP in a short time. Furthermore, it is fundamental to develop an algorithm for solving the bottleneck GAP or fuzzy GAP.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

Acknowledgments

This work was supported by the Hubei Superior and Distinctive Discipline Group of Mechatronics and Automobile (no. XKQ2020004), Innovation Scientists and Technicians Troop Construction Projects of Henan Province (CXTD2016054), and Innovative Scientists and Technicians Team of Henan Provincial High Education (no. 20IRTSTHN019).

References

[1] A. Haghani and S.-C. Oh, "Formulation and solution of a multi-commodity, multi-modal network flow model for disaster relief operations," *Transportation Research Part A: Policy and Practice*, vol. 30, no. 3, pp. 231–250, 1996.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows*, Springer, Berlin, Germany, 1988.

[3] J. Kennington and Z. Wang, "A shortest augmenting path algorithm for the semi-assignment problem," *Operations Research*, vol. 40, no. 1, pp. 178–187, 1992.

[4] M. Dell'Amico and S. Martello, "The k-cardinality assignment problem," *Discrete Applied Mathematics*, vol. 76, no. 1–3, pp. 103–121, 1997.

[5] C. Prins, "An overview of scheduling problems arising in satellite communications," *The Journal of the Operational Research Society*, vol. 45, no. 6, pp. 611–623, 1994.

[6] E. Balas and P. R. Landweer, "Traffic assignment in communication satellites," *Operations Research Letters*, vol. 2, no. 4, pp. 141–147, 1983.

[7] M. Dell'Amico, A. Lodi, and S. Martello, "Efficient algorithms and codes for k-cardinality assignment problems," *Discrete Applied Mathematics*, vol. 110, no. 1, pp. 25–40, 2001.

[8] A. Volgenant, "Solving the k-cardinality assignment problem by transformation," *European Journal of Operational Research*, vol. 157, no. 2, pp. 322–331, 2004.

[9] I. Belik, "The analysis of split graphs in social networks based on the k-cardinality assignment problem," *International Journal of Network Science*, vol. 1, no. 1, pp. 53–62, 2016.

[10] M. L. Fisher and R. Jaikumar, "A generalized assignment heuristic for vehicle routing," *Networks*, vol. 11, no. 2, pp. 109–124, 1981.

[11] P. C. Chu and J. E. Beasley, "A genetic algorithm for the generalised assignment problem," *Computers & Operations Research*, vol. 24, no. 1, pp. 17–23, 1997.

[12] E. Munapo, M. Lesaoana, P. Nyamugure, and S. Kumar, "A transportation branch and bound algorithm for solving the generalized assignment problem," *International Journal of System Assurance Engineering and Management*, vol. 6, no. 3, pp. 217–223, 2015.

[13] P. Avella, M. Boccia, and I. Vasilyev, "A computational study of exact knapsack separation for the generalized assignment problem," *Computational Optimization and Applications*, vol. 45, no. 3, pp. 543–555, 2010.

[14] M. L. Fisher, R. Jaikumar, and L. N. Van Wassenhove, "A multiplier adjustment method for the generalized assignment problem," *Management Science*, vol. 32, no. 9, pp. 1095–1103, 1986.

[15] R. M. Nauss, "Solving the generalized assignment problem: an optimizing and heuristic approach," *Informs Journal on Computing*, vol. 15, no. 3, pp. 249–266, 2003.

[16] S. Martin, "A branch-and-price algorithm for the generalized assignment problem," *Operations Research*, vol. 45, no. 6, pp. 831–841, 1997.

[17] A. J. Woodcock and J. M. Wilson, "A hybrid tabu search/branch & bound approach to solving the generalized assignment problem," *European Journal of Operational Research*, vol. 207, no. 2, pp. 566–578, 2010.

[18] J. A. Diaz and E. Fernandez, "A tabu search heuristic for the generalized assignment problem," *European Journal of Operational Research*, vol. 132, no. 1, pp. 22–38, 2001.

[19] I. H. Osman, "Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches," *Operations-Research-Spektrum*, vol. 17, no. 4, pp. 211–225, 1995.

[20] J. Majumdar and A. K. Bhunia, "An alternative approach for unbalanced assignment problem via genetic algorithm," *Applied Mathematics and Computation*, vol. 218, no. 12, pp. 6934–6941, 2012.

[21] M. Fatih Tasgetiren, P. N. Suganthan, T. J. Chua, and A. Al-Hajri, "Differential evolution algorithms for the generalized assignment problem," in *Proceedings of the 2009 IEEE Congress on Evolutionary Computation*, pp. 2606–2613, IEEE, Trondheim, Norway, May 2009.

[22] K. Sethanan and R. Pitakaso, "Improved differential evolution algorithms for solving generalized assignment problem," *Expert Systems with Applications*, vol. 45, pp. 450–459, 2016.

[23] L. Özbakir, A. Baykasoğlu, and P. Tapkan, "Bees algorithm for generalized assignment problem," *Applied Mathematics and Computation*, vol. 215, no. 11, pp. 3782–3795, 2010.

[24] S. K. Dubey, A. Kumar, and V. Upadhyay, "The average sum method for the unbalanced assignment problems," *International Journal of Mathematics Trends and Technology*, vol. 55, no. 2, pp. 89–100, 2018.

[25] Q. Rabbani, A. Khan, and A. Quddoos, "Modified Hungarian method for unbalanced assignment problem with multiple jobs," *Applied Mathematics and Computation*, vol. 361, pp. 493–498, 2019.

[26] A. Khandelwal, "An amalgamated approach for solving unbalanced assignment problem," *Malaya Journal of Matematik*, vol. 6, no. 2, pp. 321–325, 2018.

[27] M. Posta, J. A. Ferland, and P. Michelon, "An exact method with variable fixing for solving the generalized assignment problem," *Computational Optimization and Applications*, vol. 52, no. 3, pp. 629–644, 2012.

[28] H. Zhu, D. Liu, S. Zhang, Yu Zhu, L. Teng, and S. Teng, "Solving the many to many assignment problem by improving the Kuhn-Munkres algorithm with backtracking," *Theoretical Computer Science*, vol. 618, pp. 30–41, 2016.

[29] Y. Hu, X. Zhao, J. Liu, B. Liang, and C. Ma, "An efficient algorithm for solving minimum cost flow problem with complementarity slack conditions," *Mathematical Problems in Engineering*, vol. 2020, Article ID 2439265, 2020.

[30] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice-Hall, Upper Saddle River, NJ, USA, 1993.

[31] W. Boonphakdee and P. Charnsethikul, "Solving large scale assignment problem using the successive complementary slackness conditions," in *Proceedings of the 2nd International Conference on Mathematics, Engineering and Industrial Applications 2016*, Songkhla, Thailand, August 2016.

[32] R. Jonker and A. Volgenant, "A shortest augmenting path algorithm for dense and sparse linear assignment problems," *Computing*, vol. 38, no. 4, pp. 325–340, 1987.

[33] R. Jonker and A. Volgenant, "Linear assignment procedures," *European Journal of Operational Research*, vol. 116, no. 1, pp. 233-234, 1999.

[34] D. Knuth, "The art of computer programming," in *Seminumerical Algorithms*vol. 2, 2nd edition, 1981.