International Journal of Industrial Engineering Computations 13 (2022) 267–276

Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

An exact algorithm for constrained *k*-cardinality unbalanced assignment problem

A. Prakash[a], Uruturu Balakrishna[b*] and Jayanth Kumar Thenepalle[c]

[a]*Research Scholar, Department of Mathematics, JNTUA Anantapurammu, India*
[b]*Professor, Department of Science & Humanities,Sreeenivasa Institute of Technology and Management Studes, Chittoor, India*
[c]*Assistant Professor, Department of Science & Humanities, Sreeenivasa Institute of Technology and Management Studes, India*

C H R O N I C L E

Article history:
Received May 11 2021
Received in Revised Format June 28 2021
Accepted October 22 2021
Available online October, 22 2021

Keywords:
k-cardinality
Unbalanced assignment problem
Zero-one integer programming
Lexi-search algorithm

A B S T R A C T

An assignment problem (AP) usually deals with how a set of persons/tasks can be assigned to a set of tasks/persons on a one-to-one basis in an optimal manner. It has been observed that balancing among the persons and jobs in several real-world situations is very hard, thus such scenarios can be seen as unbalanced assignment models (UAP) being a lack of workforce. The solution techniques presented in the literature for solving UAP's depend on the assumption to allocate some of the tasks to fictitious persons; those tasks assigned to dummy persons are ignored at the end. However, some situations in which it is inevitable to assign more tasks to a single person. This paper addresses a practical variant of UAP called *k*-cardinality unbalanced assignment problem (*k*-UAP), in which only of persons are asked to perform jobs and all the persons should perform at least one and at most jobs. The *k*-UAP aims to determine the optimal assignment between persons and jobs. To tackle this problem optimally, an enumerative Lexi-search algorithm (LSA) is proposed. A comparative study is carried out to measure the efficiency of the proposed algorithm. The computational results indicate that the suggested LSA is having the great capability of solving the smaller and moderate instances optimally.

1. Introduction

The classical assignment problem (AP) is an exceptional case of Transportation problem that aims to seek an optimal schedule between persons (m) and jobs (n) with one-to-one correspondence. Votaw and Orden (1952) however first discussed the concept of assignment problem; the Hungarian method by Kuhn (1955) is the first and significant solution technique for solving AP. Generally, an AP with $m = n$ is known as the balanced assignment problem or simply an assignment problem. However, in most of the practical scenarios, the number of persons and jobs may not be the same $(\text{i.e.} m \neq n)$. A problem of this kind is called unbalanced assignment problem (UAP). Based on the number of persons and jobs involved, the problems can be further classified into two types, namely

(i) UAP with $m < n$
(ii) UAP with $m > n$

Due to the scarcity of workforce, several UAP models can be viewed as problems of the first type $(\text{i.e.} m < n)$. These kinds of problems are generally tackled using Hungarian methods and converting unbalanced assignment models into balanced ones by just deploying dummy/ fictitious persons for the first type of problems and dummy/ fictitious jobs for the second type of models. Thus, for the problem of the first kind, few jobs remain idle whereas few persons remain unassigned for the second type of problems. Some of the persons or jobs remain unassigned and thus they persist idle for both kinds of problems. As a result, any organization attains some loss due to not utilizing the resources effectively. Therefore, it is inevitable to address the UAB scenarios in which no persons or jobs are to be left idle.

* Corresponding author
E-mail: prasantibalu@gmail.com (U. Balakrishna)

2022 Growing Science Ltd.
doi: 10.5267/j.ijiec.2021.10.002

Reviewing the literature, Malhotra and Bhatia (1984) addressed two practical variants AP's, where the first one is UAP with $m > n$ and the second one is the UAP with an additional constraint on the minimum number of jobs to be executed by each person. Arora and Puri (1998) studied time minimizing assignment problems in which each job should be performed by precisely a single person and each person can do more than one job with an assumption that all the persons can execute the jobs at the same time. To solve this problem optimally, an exact LSA has been proposed. Kumar (2006) has developed a modified Hungarian method to solve an UAP in which the number of jobs is more than the machines. This modified method effectively assigns the given number of jobs to the machines. Iampang et. al., (2010) considered the UAP with $m > n$ and solves by using a cost and space-efficient methods. This study showed that the solution obtained by cost and space-efficient method is better than that of Kumar's method discussed earlier. Majumdar and Bhunia (2012) presented an efficient Genetic algorithm (GA) for solving UAP with $m < n$. Yadaiah and Haragopal (2016) developed the Lexi-search algorithm for solving UAP with $m > n$. Bhunia et. al., (2017) first considered the UAP with $m < n$ over interval costs together with an additional condition over the number of jobs permitted to do by each person. A GA based solution method is developed to solve this problem effectively. Recently, Thenepalle and Singamsetty (2019) addressed UAP with multiple dimensions and two objectives, where the first objective minimizes the overall time to perform the jobs and the second objective maximizes the overall production on executing the tasks. This model also includes an additional condition over the number of jobs that a person is permitted to perform. To tackle this problem, an exact LSA is developed, which effectively provides efficient Pareto optimal solutions. The works cited above show the significance and scope of UAP with additional practical constraints.

Several variants have been addressed in the literature as AP has practical utility. The *k*-cardinality assignment problem (*k*-AP) is one of the variants of the classical AP, which was first introduced by Dell'Amico and Martello (1997). The so-called *k*-AP can be defined as: Let there are m persons and n jobs, whose cost matrix for performing each job by a person is given by $C = (c_{ij})_{m\times n}, c_{ij} \geq 0$. Let a positive integer k where $k \leq m$. The *k*-AP aims to select k persons and k jobs from m persons and n jobs, respectively such that the overall cost incurred is minimized on performing k jobs by k persons. The model *k*-AP has several potential applications namely satellite communication and switching problems (Dell'Amico and Martello, 1997), production planning (Gabrovšek et. al., 2020) etc. With its wide applicability, several researchers have considered *k*-AP and developed various solution methods. For the development of *k*-AP, we may see the works (Dell'Amico and Martello, 1997; Dell'Amico et. al., 2001; Volgenant, 2004; Feng and Yang, 2006; Bai, 2009; Belik and Jörnsten, 2016). However, all the above-cited works on *k*-AP looks for choosing k persons and k jobs from m persons and n jobs, respectively, which still seem a balanced *k*-AP. The models of *k*-cardinality unbalanced assignment problem (*k*-UAP) with some practical constraints are still open. In addition to the *k*-AP, many researchers have applied the cardinality constraint in various combinatorial optimization problems such as TSP (Bhavani and Murthy, 2006), Minimum spanning tree problem (Kumar and Purusotham, 2017 & 2018), Multi-objective TSP (Thenepalle and Singamsetty, 2018) etc.

The above-cited works motivate to attempt a practical variant of UAP called *k*-cardinality unbalanced assignment problem with additional constraint (*k*-UAP). This study also includes a cardinality constraint over the number of persons and an additional condition over number of jobs that is to be done by each person. To deal with this problem optimally, an exact LSA is proposed.

The remaining of the paper is organized as follows: A detailed description of the *k*-UAP and its formulation is given in Section 2. Section 3 describes the preliminaries and the steps involved in the proposed LSA. Section 4 reports computational results. Finally, the conclusions and scope of future work are given in Section 5.

2. Mathematical Formulation

Let the sets $I = \{1,2,3,..m\}$ and $J = \{1,2,3,..n\}$ consists of m persons and n jobs, respectively with $m < n$. Let k be a non-negative integer such that $k \leq m$. The objective coefficient c_{ij} (the cost incurred on executing j^{th} job by i^{th} person) is defined on each $(i, j) \forall\ i \in I, j \in J$. Further, an additional condition over n that a person is permitted to execute is also considered. The problem *k*-UAP seeks to find the optimal assignment of performing n jobs by k out of m persons such that each job is to be given to just one person and a person is permitted to execute multiple jobs.

2.1. Assumptions

a). Number of persons is less than the number of jobs $(i.e.\ m < n)$.
b). n jobs are assigned to k out of m persons.
c). Each job is executed by just one person.
d). $m - k$ persons remain unassigned
e). k persons can do at least one and atmost $n - k + 1$ jobs.
f). All the jobs are started simultaneously.

The mathematical model of proposed *k*-UAP using zero-one integer linear programming (0-1 ILP) is as follows:

$$\min Z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{1}$$

subject to

$$\sum_{i=1}^{m} x_{ij}=1,\ \forall\, j \in J \tag{2}$$

$$\sum_{j=1}^{n} x_{ij}\geq 0,\ \forall\, i \in I \tag{3}$$

$$\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij} = n \tag{4}$$

$$0 \leq \sum_{j=1}^{n} x_{ij} \leq n-k+1,\ \forall\, i \in I \tag{5}$$

$$x_{ij} = \{1,0\},\ \forall\, i \in I, j \in J \tag{6}$$

Here, total processing cost/time on performing the jobs is minimized through the objective function (1). The Constraint (2) assures that each job should be performed by just one person. The Constraint (3) represents out of m persons, $m-k$ persons remain unassigned whereas the k persons need to do more than or equals to one job. The Constraint (4) ensures that all the given n jobs should be performed. Constraint (5) ensures each of the k persons is allowed to perform at least one and at most $n-k+1$ jobs and remaining persons will not be assigned to any job. Finally, the binary variable x_{ij} in Constraint (6) takes 1 when i^{th} person performs a j^{th} job and 0, otherwise.

3. Lexi-search Algorithm: Preliminaries

3.1. Pattern

A pattern is usually a two-dimensional matrix *X* that refers the feasible assignment. The value of the pattern *X* is computed using the following formula.

$$V(X) = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}\, x_{ij} \tag{7}$$

3.2. Alphabet table

The cost matrix $C = [c_{ij}]_{m\times n}$ consists mn ordered pairs. The elements of C are organized in ascending sequence and marked from 1 to mn, saved in an array SN (Sundara Murthy, 1976). If $a_1, a_2 \in SN$ and $a_1 < a_2$ then $C(a_1) \leq C(a_2)$. Let $L_r = (a_1, a_2, ..., a_r)$, $a_i \in SN$ be a string of r indices. The indices in L_r are organized in such a way that $a_i < a_{i+1}$ $\forall\ i = 1,2,...,r-1$. The word L_r with $a_i < a_{i+1}$ $\forall\ i = 1,2,...,r-1$ is termed as 'sensible word' and otherwise, it is referred to as 'non-sensible word'. The systematic sequence of r indices from SN is known as a 'word' of length r. The word L_r of length $r < n$ is said to be the partial feasible word, whereas the word L_r having length $r = n$ is called a complete feasible word.

3.3. Lexi-search mechanism

The Lexi-search algorithm (LSA) is a systematized Branch & Bound (B&B) approach, proposed by Pandit (1962) for resolving the loading problem. This approach is proved to have a great capability in solving many combinatorial optimization problems optimally (Pandit,1962).

"It is possible to list out all the solutions in a systematic hierarchy, which also have a hierarchical ordering of the corresponding values of the solutions".

The performance of the LSA usually be governed by on the choice of the suitable alphabet table, which can be done in two ways (Sundara Murthy, 1976).

"The process of checking the feasibility of a partial word is easy; on the other hand, the calculation of lower bound is complex".

"The computation of lower bound is easy; while the feasibility checking is difficult".

3.4. Bounds Calculation

As the objective function *Z* is of minimization type, the upper bound (*TS=UB*) of *Z* is supposed to have a very large value. The lower bound *LB* and the value of *V* for a partial word L_r is determined as follows:

$LB(L_r) = V(L_r) + CT(a_r + n - r) - CT(a_r)$; where $V(L_r) = V(L_{r-1}) + C(a_r)$ with $V(L_0) = 0$

3.5. Proposed LSA

The step by step process of proposed LSA for *k-UAP* is illustrated below.

Step-1: *Initialization*

k and cost matrix $(C = [c_{ij}])$.

Set $TS = UB = 9999$.

Step-2: If the problem dimension fulfills the condition $m < n$ and $k \le m$, then move to Step 3; else, no feasible solution for the problem and move to Step-14.

Step-3: Generate the alphabet table for the cost matrix (C). Move to Step-4.

Step-4: The search begins with $L_r = (a_1) = 1$. Move to Step-5.

Step-5: Calculate $LB(L_r)$. If $LB(L_r) < TS$ then go to Step-6; else, move to Step-10.

Step-6: If L_r is feasible, then we admit it and remain for next partial word of order $r+1$ and go to Step-7; else, consider the successive partial word of order r by taking another letter that succeeds a_r in its r^{th} position and move to Step-5.

Step-7: If $r = n$ then go to Step-8; else, go to Step-9.

Step-8: Update TS by $LB(L_r)$ and move to Step-11.

Step-9: The partial word L_r can be concatenated by using $L_r = L_{r-1} \otimes (a_r)$ where $\otimes$ defines the concatenation operation and go to Step-5.

Step-10: If $r = 1$, then go to Step-14; else move to Step-11.

Step-11: Take $TS = UB$ and proceed further by taking $r = r - 1$ and go to Step-5.

Step-12: Continue the Steps 5-11 sequentially till TS has no further development and go to Step-13.

Step-13: Record TS. Move to Step-14.

Step-14: Stop

Finally, the currrent TS gives the optimal solution.

4. Numerical Example

To demonstrate the proposed LSA, a suitable example with five persons (*m=5*), seven jobs (*n=7*) and cardinality over number of persons is restricted to two (*k=2*) is considered. The person-job assignment matrix $C=[c_{ij}]$ is provided in Table 1.

Table 1
The cost of performing every job by each person

	J_1	J_2	J_3	J_4	J_5	J_6	J_7
P_1	21	12	2	26	21	7	18
P_2	9	10	5	22	25	12	20
P_3	15	24	30	16	29	21	15
P_4	19	28	11	32	27	17	26
P_5	14	23	26	2	3	4	3

The alphabet table for the cost matrix $C=[c_{ij}]$ is provided in Table 2. In Table 2, the the array notations SN, Ct, CCt, R, C indicates Index number, values of cost matrix in ascending order, cumulative cost, row, column indices, respectively.

Table 2
Alphabet table

SN	*Ct*	*CCt*	*R*	*C*	*SN*	*Ct*	*CCt*	*R*	*C*
1	2	2	1	3	19	19	194	4	1
2	2	4	5	4	20	20	214	2	7
3	3	7	5	5	21	21	235	1	1
4	3	10	5	7	22	21	256	1	5
5	4	14	5	6	23	21	277	3	6
6	5	19	2	3	24	22	299	2	4
7	7	26	1	6	25	23	322	5	2
8	9	35	2	1	26	24	346	3	2
9	10	45	2	2	27	25	371	2	5
10	11	56	4	3	28	26	397	1	4
11	12	68	1	2	29	26	423	4	7
12	12	80	2	6	30	26	449	5	3
13	14	94	5	1	31	27	476	4	5
14	15	109	3	1	32	28	504	4	2
15	15	124	3	7	33	29	533	3	5
16	16	140	3	4	34	30	563	3	3
17	17	157	4	6	35	32	595	4	4
18	18	175	1	7					

The logical flow of the proposed LSA is presented in Table 3. The column *SN* in Table 3 denotes the index number. As the total jobs done by two persons is seven, thus the length of the word L_r becomes 7. Hence, the columns after *SN* are denoted by 1, 2, 3, 4, 5, 6, 7, each one indicating the letters in corresponding places of a word. The next two columns *Vc & LB* represent the value and lower bound of a partial word L_r, respectively. The subsequent two columns *R* and *C,* respectively denotes the row and column indices of the respective letter. Finally, the last column *Remark* represents accept the partial word when it is feasible and marked as *Ac* i.e Accept; otherwise R_j i.e reject. Initially, the algorithm starts by assuming a trial solution *TS=999999*, a large value. For each *SN*, *LB* of a partial word is evaluated and confirmed if it fulfills the bounds or not. A partial word is marked Ac only when the *LB* is less than the trial solution and satisfies the constraints given in Section 3. A partial word is marked *Rj* when it fails to satisfy any one of the constraints given in Section 3. Similarly, if the *LB*≥*TS*, then it is rejected and denoted by ≥*TS, Rj*. In Table 3, it is seen that the initial feasible solution (*TS = 40*) is found at 13th row and the respective feasible word is L_7*=(1,2,3,4,5,11,13)*.

Table 3
Logical flow of proposed LSA

SN	1	2	3	4	5	6	7	*V*	*LB*	*R*	*C*	*Remark*
1	1							2	26	1	3	*Ac*
2		2						4	26	5	4	*Ac*
3			3					7	26	5	5	*Ac*
4				4				10	26	5	7	*Ac*
5					5			14	26	5	6	*Ac*
6						6		19	26	2	3	*Rj*
7						7		21	30	1	6	*Rj*
8						8		23	33	2	1	*Rj*
9						9		24	35	2	2	*Rj*
10						10		25	37	4	3	*Rj*
11						11		26	38	1	2	*Ac*
12							12	38	38	2	6	*Rj*
13							13	40	40	5	1	*A,TS = 40*
14						12		26	40	2	6	*≥TS, Rj*
15					6			15	31	2	3	*Rj*
16					7			17	36	1	6	*Ac*
17						8		26	36	2	1	*Rj*
18						9		27	38	2	2	*Rj*
19						10		28	40	4	3	*≥TS, Rj*
20					8			19	40	2	1	*≥TS, Rj*
21				5				11	32	5	6	*Ac*
22					6			16	32	2	3	*Rj*
23					7			18	37	1	6	*Rj*
24					8			20	41	2	1	*≥TS, Rj*
25				6				12	38	2	3	*Rj*

Table 3
Logical flow of proposed LSA (Continued)

SN	1	2	3	4	5	6	7	*V*	*LB*	*R*	*C*	*Remark*
26				7				14	44	1	6	*≥TS, Rj*
27			4					7	32	5	7	*Ac*
28				5				11	32	5	6	*Ac*
29					6			16	32	2	3	*Rj*
30					7			18	37	1	6	*Rj*
31					8			20	41	2	1	*≥TS, Rj*
32				6				12	38	2	3	*Rj*
33				7				14	44	1	6	*≥TS, Rj*
34			5					8	39	5	6	*Ac*
35				6				13	39	2	3	*Rj*
36				7				15	45	1	6	*≥TS, Rj*
37			6					9	46	2	3	*≥TS, Rj*
38		3						5	33	5	5	*Ac*
39			4					8	33	5	7	*Ac*
40				5				12	33	5	6	*Ac*
41					6			17	33	2	3	*Rj*
42					7			19	38	1	6	*Rj*
43					8			21	42	2	1	*≥TS, Rj*
44				6				13	39	2	3	*Rj*
45				7				15	45	1	6	*≥TS, Rj*
46			5					9	40	5	6	*≥TS, Rj*
47		4						5	40	5	7	*≥TS, Rj*
48	2							2	33	5	4	*Ac*
49		3						5	33	5	5	*Ac*
50			4					8	33	5	7	*Ac*
51				5				12	33	5	6	*Ac*
52					6			17	33	2	3	*Ac*
53						7		24	33	1	6	*Rj*
54						8		26	36	2	1	*Ac*
55							9	26	36	2	2	*Ac, TS =36*
56						9		27	37	2	2	*≥TS, Rj*
57					7			19	38	1	6	*≥TS, Rj*
58				6				13	39	2	3	*≥TS, Rj*
59			5					9	40	5	6	*≥TS, Rj*
60		4						5	40	5	7	*≥TS, Rj*
61	3							3	41	5	5	*≥TS, Rj*

To improve the solution, a backtracking strategy is applied and the next improved solution (*TS=36*) is obtained at 55th row of the Table 3, whose feasible word is L_7*=(2,3,4,5,8,9).* Since there is no further improvement of the current solution (*VT=36*), the search process completes at 61st row and the latest found solution (*VT=36*) is considered as the optimal solution. The full-length words of feasible and optimal solutions and their corresponding schedules are reported in Table 4.

Table 4
Feasible and optimal schedules

SN	Full-length word	Feasible/Optimal Schedule	Feasible/ Optimal Solution
1	L_7 = (1,2,3,4,5,11,13)	(1,3), (5,4), (5,5), (5,7), (5,6), (1,2), (5,1)	40
2	L_7 = (2,3,4,5,6,8,9)	(5,4), (5,5), (5,7), (5,6), (2,3), (2,1), (2,2)	36

The feasible and optimal schedules reported in Table 4 are also represented as $P_1 \to J_2 \wedge J_3$, $P_5 \to J_1 \wedge J_4 \wedge J_5 \wedge J_6 \wedge J_7$ and $P_2 \to J_1 \wedge J_2 \wedge J_3$, $P_5 \to J_4 \wedge J_5 \wedge J_6 \wedge J_7$ respectively. For better understanding $P_1 \to J_2 \wedge J_3$ indicates that second and third jobs are performed by first person. Similarly, $P_5 \to J_4 \wedge J_5 \wedge J_6 \wedge J_7$ denotes fourth, fifth, sixth and seventh jobs are done by fifth person.

5. Computational Results

All the experiments were tested in Matlab 2017a, Intel Core i5 2.10 with GHz CPU B950 and 4 GB of RAM PC running Microsoft Windows 2010 OS. The computational results including comparative results of relaxed version of *k-UAP* and computational results of *k-UAP* using proposed LSA tested on some selected test instances as used by Majumdar & Bhunia (2012). To compare the results of the proposed LSA against the results of Majumdar & Bhunia (2012), the current model *k-UAP* will transform to an equivalent model addressed by Majumdar & Bhunia (2012) by setting $k = m$ and imposing a constraint over the jobs that are permitted to do by each person.

5.1 Comparision of proposed LSA and Genetic Algorithm of Majumdar & Bhunia (2012)

The proposed LSA performance can be assessed through a comparative study of the genetic algorithm (GA) developed by Majumdar and Bhunia (2012) for UAP with $m < n$, whose best GA results (GA^*) are considered for comparison purposes. These results are acquired by testing two distinct variants of GA (GA-1 and GA-2) over the cases of sizes 5 × 7, 5 × 8, 6 × 10

and 7 × 10 with distinct parametric values. Note that these instances are denoted as AP-1, AP-2, AP-3 and AP-4, respectively. All these instances are also provided in the Appendix. Tables 5-7 provide the comparative details of LSA and GA*.

Table 5
Comparative results of LSA against GA* for AP-1 and AP-2

Algorithm	Instances→	AP-1	AP-1	AP-2	AP-2	AP-2
GA*	Jobs	2	3	2	3	4
	Worst Solution	137	136	1530	1510	1500
	Optimal Solution	133	128	1520	1470	1450
	Best Runtime	0.001	0.001	0.001	0.001	0.001
	Optimal Solution Corresponding Schedule& Alternate Schedule (@)	$P_1 \rightarrow J_2 \wedge J_3$ $P_2 \rightarrow J_4$ $P_3 \rightarrow J_5$ $P_4 \rightarrow J_7$ $P_5 \rightarrow J_1 \wedge J_6$ @$P_1 \rightarrow J_3 \wedge J_6$ $P_2 \rightarrow J_4$ $P_3 \rightarrow J_5$ $P_4 \rightarrow J_7$ $P_5 \rightarrow J_1 \wedge J_2$	$P_1 \rightarrow J_3$ $P_2 \rightarrow J_4$ $P_3 \rightarrow J_5$ $P_4 \rightarrow J_7$ $P_5 \rightarrow J_1 \wedge J_2 \wedge J_6$	$P_1 \rightarrow J_2 \wedge J_3$ $P_2 \rightarrow J_4 \wedge J_8$ $P_3 \rightarrow J_5$ $P_4 \rightarrow J_7$ $P_5 \rightarrow J_1 \wedge J_6$ @$P_1 \rightarrow J_3 \wedge J_6$ $P_2 \rightarrow J_4 \wedge J_8$ $P_3 \rightarrow J_5$ $P_4 \rightarrow J_7$ $P_5 \rightarrow J_1 \wedge J_2$	$P_1 \rightarrow J_3$ $P_2 \rightarrow J_4 \wedge J_8$ $P_3 \rightarrow J_5$ $P_4 \rightarrow J_7$ $P_5 \rightarrow J_1 \wedge J_2 \wedge J_6$	$P_1 \rightarrow J_3$ $P_2 \rightarrow J_8$ $P_3 \rightarrow J_4$ $P_4 \rightarrow J_7$ $P_5 \rightarrow J_1 \wedge J_2 \wedge J_5 \wedge J_6$
LSA	Jobs	2	3	2	3	4
	Worst Solution	137	140	1560	1590	1580
	Optimal Solution	133	128	1520	1470	1450
	Best Runtime	0.010	0.010	0.010	0.010	0.010
	Optimal Solution Corresponding Schedule & Alternate Schedule (@)	$P_1 \rightarrow J_2 \wedge J_3$ $P_2 \rightarrow J_4$ $P_3 \rightarrow J_5$ $P_4 \rightarrow J_7$ $P_5 \rightarrow J_1 \wedge J_6$ @$P_1 \rightarrow J_3$ $P_2 \rightarrow J_4$ $P_3 \rightarrow J_5$ $P_4 \rightarrow J_6 \wedge J_7$ $P_5 \rightarrow J_1 \wedge J_2$	$P_1 \rightarrow J_3$ $P_2 \rightarrow J_4$ $P_3 \rightarrow J_5$ $P_4 \rightarrow J_7$ $P_5 \rightarrow J_1 \wedge J_2 \wedge J_6$	$P_1 \rightarrow J_3 \wedge J_6$ $P_2 \rightarrow J_4 \wedge J_8$ $P_3 \rightarrow J_5$ $P_4 \rightarrow J_7$ $P_5 \rightarrow J_1 \wedge J_2$ @$P_1 \rightarrow J_3$ $P_2 \rightarrow J_4 \wedge J_8$ $P_3 \rightarrow J_5$ $P_4 \rightarrow J_6 \wedge J_7$ $P_5 \rightarrow J_1 \wedge J_2$	$P_1 \rightarrow J_3$ $P_2 \rightarrow J_4 \wedge J_8$ $P_3 \rightarrow J_5$ $P_4 \rightarrow J_7$ $P_5 \rightarrow J_1 \wedge J_2 \wedge J_6$	$P_1 \rightarrow J_3$ $P_2 \rightarrow J_8$ $P_3 \rightarrow J_4$ $P_4 \rightarrow J_7$ $P_5 \rightarrow J_1 \wedge J_2 \wedge J_5 \wedge J_6$

In Table 5, for all the instances of AP-1 and AP-2, the best results of LSA match with GA*. The CPU run time of LSA varies from 0.010 to 0.014 seconds, whereas GA* converges to 0.001 seconds. In Table 6 and Table 7, the best solutions acheived through LSA match with GA* for 4 cases, such as AP-3 (with 2 and 4 jobs) and AP-4 (with 2 and 3 jobs).

Table 6
Comparative results of LSA against GA* for AP-3

Algorithm	Instances→	AP-3	AP-3	AP-3	AP-3
GA*	Jobs	2	3	4	5
	Worst Solution	80	75	70	78
	Optimal Solution	66	67	66	74
	Best Runtime	0.060	0.060	0.060	0.060
	Optimal Solution Corresponding Schedule& Alternate Schedule(@)	$P_1 \rightarrow J_2 \wedge J_5$ $P_2 \rightarrow J_4 \wedge J_7$ $P_3 \rightarrow J_1 \wedge J_3$ $P_4 \rightarrow J_6$ $P_5 \rightarrow J_8$ $P_6 \rightarrow J_9 \wedge J_{10}$	$P_1 \rightarrow J_2 \wedge J_5$ $P_2 \rightarrow J_4 \wedge J_6 \wedge J_7$ $P_3 \rightarrow J_1 \wedge J_3$ $P_4 \rightarrow J_{10}$ $P_5 \rightarrow J_8$ $P_6 \rightarrow J_9$	$P_1 \rightarrow J_2$ $P_2 \rightarrow J_4 \wedge J_5 \wedge J_6 \wedge J_7$ $P_3 \rightarrow J_1 \wedge J_3$ $P_4 \rightarrow J_{10}$ $P_5 \rightarrow J_8$ $P_6 \rightarrow J_9$	$P_1 \rightarrow J_2$ $P_2 \rightarrow J_4 \wedge J_5 \wedge J_6 \wedge J_7 \wedge J_{10}$ $P_3 \rightarrow J_1$ $P_4 \rightarrow J_3$ $P_5 \rightarrow J_8$ $P_6 \rightarrow J_9$
LSA	Jobs	2	3	4	5
	Worst Solution	72	71	68	71
	Optimal Solution	66	65	66	69
	Best Runtime	0.010	0.010	0.010	0.012
	Optimal Solution Corresponding Schedule & Alternate Schedule(@)	$P_1 \rightarrow J_2 \wedge J_5$ $P_2 \rightarrow J_4 \wedge J_7$ $P_3 \rightarrow J_1 \wedge J_3$ $P_4 \rightarrow J_6$ $P_5 \rightarrow J_8$ $P_6 \rightarrow J_9 \wedge J_{10}$	$P_1 \rightarrow J_2$ $P_2 \rightarrow J_4 \wedge J_5 \wedge J_7$ $P_3 \rightarrow J_1 \wedge J_3$ $P_4 \rightarrow J_6$ $P_5 \rightarrow J_8$ $P_6 \rightarrow J_9 \wedge J_{10}$	$P_1 \rightarrow J_2$ $P_2 \rightarrow J_4 \wedge J_5 \wedge J_6 \wedge J_7$ $P_3 \rightarrow J_1 \wedge J_3$ $P_4 \rightarrow J_{10}$ $P_5 \rightarrow J_8$ $P_6 \rightarrow J_9$	$P_1 \rightarrow J_2$ $P_2 \rightarrow J_3 \wedge J_4 \wedge J_5 \wedge J_6 \wedge J_7$ $P_3 \rightarrow J_1$ $P_4 \rightarrow J_{10}$ $P_5 \rightarrow J_8$ $P_6 \rightarrow J_9$ @ $P_1 \rightarrow J_2$ $P_2 \rightarrow J_1 \wedge J_4 \wedge J_5 \wedge J_6 \wedge J_7$ $P_3 \rightarrow J_3$ $P_4 \rightarrow J_{10}$ $P_5 \rightarrow J_8$ $P_6 \rightarrow J_9$

For remaining cases, specifically AP-3 (with 3 and 5 jobs) and AP-4 (with 4 jobs) LSA provided the preferable results than the GA*. The computational run time of LSA ranges from 0.010 to 0.015 seconds, whereas GA* ranges from 0.060 to 0.120 seconds. Here, LSA seems more competent than GA* as per with solution quality and runtime aspects. It is evident that the

LSA is also list out the alternative solutions efficiently, if it occurs. The other best found solutions can also be seen in Tables 5-7. This indicates the ability of LSA to list out all the possible alternative solutions. The overall results show that the LSA is outperforming the GA* in both quality of the solution and runtime of the algorithm aspects.

Table 7
Comparative results of LSA against GA* for AP-4

Algorithm	Instances→	AP-4	AP-4	AP-4
GA*	Jobs	2	3	4
	Worst Solution	71	75	82
	Optimal Solution	69	69	73
	Best Runtime	0.120	0.100	0.110
	Optimal Solution	$P_1 \rightarrow J_4$	$P_1 \rightarrow J_4$	$P_1 \rightarrow J_4$
	Corresponding	$P_2 \rightarrow J_6 \wedge J_7$	$P_2 \rightarrow J_6$	$P_2 \rightarrow J_{10}$
	Schedule& Alternate	$P_3 \rightarrow J_1$	$P_3 \rightarrow J_{10}$	$P_3 \rightarrow J_1$
	Schedule(@)	$P_4 \rightarrow J_3 \wedge J_8$	$P_4 \rightarrow J_3 \wedge J_7 \wedge J_8$	$P_4 \rightarrow J_3 \wedge J_6 \wedge J_7 \wedge J_8$
		$P_5 \rightarrow J_9$	$P_5 \rightarrow J_9$	$P_5 \rightarrow J_9$
		$P_6 \rightarrow J_5$	$P_6 \rightarrow J_1 \wedge J_5$	$P_6 \rightarrow J_5$
		$P_7 \rightarrow J_2 \wedge J_{10}$	$P_7 \rightarrow J_2$	$P_7 \rightarrow J_2$
		@ $P_1 \rightarrow J_4$	@ $P_1 \rightarrow J_4$	
		$P_2 \rightarrow J_6 \wedge J_7$	$P_2 \rightarrow J_7$	
		$P_3 \rightarrow J_{10}$	$P_3 \rightarrow J_{10}$	
		$P_4 \rightarrow J_3 \wedge J_8$	$P_4 \rightarrow J_3 \wedge J_6 \wedge J_8$	
		$P_5 \rightarrow J_9$	$P_5 \rightarrow J_9$	
		$P_6 \rightarrow J_1 \wedge J_5$	$P_6 \rightarrow J_1 \wedge J_5$	
		$P_7 \rightarrow J_2$	$P_7 \rightarrow J_2$	
LSA	Jobs	2	3	4
	Worst Solution	69	69	83
	Optimal Solution	69	69	71
	Best Runtime	0.010	0.011	0.012
	Optimal Solution	$P_1 \rightarrow J_4$	$P_1 \rightarrow J_4$	$P_1 \rightarrow J_4$
	Corresponding	$P_2 \rightarrow J_6$	$P_2 \rightarrow J_6$	$P_2 \rightarrow J_6$
	Schedule & Alternate	$P_3 \rightarrow J_7 \wedge J_{10}$	$P_3 \rightarrow J_{10}$	$P_3 \rightarrow J_{10}$
	Schedule(@)	$P_4 \rightarrow J_3 \wedge J_8$	$P_4 \rightarrow J_3 \wedge J_7 \wedge J_8$	$P_4 \rightarrow J_3 \wedge J_5 \wedge J_7 \wedge J_8$
		$P_5 \rightarrow J_9$	$P_5 \rightarrow J_9$	$P_5 \rightarrow J_9$
		$P_6 \rightarrow J_1 \wedge J_5$	$P_6 \rightarrow J_1 \wedge J_5$	$P_6 \rightarrow J_1$
		$P_7 \rightarrow J_2$	$P_7 \rightarrow J_2$	$P_7 \rightarrow J_2$
		@ $P_1 \rightarrow J_4$	@ $P_1 \rightarrow J_4$	
		$P_2 \rightarrow J_6 \wedge J_7$	$P_2 \rightarrow J_7$	
		$P_3 \rightarrow J_{10}$	$P_3 \rightarrow J_{10}$	
		$P_4 \rightarrow J_3 \wedge J_8$	$P_4 \rightarrow J_3 \wedge J_6 \wedge J_8$	
		$P_5 \rightarrow J_9$	$P_5 \rightarrow J_9$	
		$P_6 \rightarrow J_1 \wedge J_5$	$P_6 \rightarrow J_1 \wedge J_5$	
		$P_7 \rightarrow J_2$	$P_7 \rightarrow J_2$	

5.2 Computational results of k-UAP using proposed LSA

This section reports the computational details of *k-UAP* tested using the proposed LSA. For computational experiments, we have considered the same four test instances as used by Majumdar and Bhunia (2012). Each of these four instances with distinct k values were tested using the proposed LSA. Each test instance is executed in ten independent runs with distinct TS values and the summary of best found results are reported in Table 8. Table 8 consists of the results for each instance such as the worst, optimal solutions, CPU runtime (in seconds) required to get the optimal solution and schedule corresponding to the optimal solution.

From Table 8, the key remarks are observed:

i. The proposed LSA produces new best solutions that may help to the future comparative purposes.
ii. The proposed LSA is proficient in achieving the substitute solutions, if exists.
iii. It is note that the test instance AP-2 with *k=2* could take 0.031 seconds while AP-2 with *k=4* has taken 0.043 seconds to get optimal solution. Similarly, AP-3 with *k=3* took 0.042 seconds whereas the test instance AP-3 with with *k=5* has used 0.028 seconds to get the optimal solution. This clearly demonstrations the performance of the LSA depends on k values.
iv. The runtime ranging from 0.027 seconds to 0.050 seconds show that the LSA takes less time for getting best solutions.

To conclude, the LSA appears to be more capable in providing promising solutions within considerably less time.

Table 8
Computational results of *k-UAP* using LSA on AP-1, AP-2, AP-3 & AP-4

Algorithm	Instances→	AP-1	AP-1	AP-1	AP-2
LSA	*k*	2	3	4	2
	Worst solution	122	129	139	1400
	Optimal solution	122	123	125	1400
	Best runtime (in Sec.)	0.029	0.038	0.044	0.031
	Optimal solution corresponding schedule & alternate schedule(@)	$P_1 \rightarrow J_3$ $P_5 \rightarrow J_1 \wedge J_2 \wedge J_4 \wedge J_5 \wedge J_6 \wedge J_7$	$P_1 \rightarrow J_3$ $P_2 \rightarrow J_4$ $P_5 \rightarrow J_1 \wedge J_2 \wedge J_5 \wedge J_6 \wedge J_7$	$P_1 \rightarrow J_3$ $P_2 \rightarrow J_4$ $P_4 \rightarrow J_7$ $P_5 \rightarrow J_1 \wedge J_2 \wedge J_5 \wedge J_6$	$P_1 \rightarrow J_3$ $P_5 \rightarrow J_1 \wedge J_2 \wedge J_4 \wedge J_5 \wedge J_6 \wedge J_7$
	Instances→	AP-2	AP-2	AP-3	AP-3
	k	3	4	2	3
	Worst solution	1470	1570	86	78
	Optimal solution	1410	1430	76	68
	Best runtime (in Sec.)	0.028	0.043	0.042	0.042
	Optimal solution corresponding schedule & alternate schedule(@)	$P_1 \rightarrow J_3$ $P_2 \rightarrow J_8$ $P_5 \rightarrow J_1 \wedge J_2 \wedge J_4 \wedge J_5 \wedge J_6 \wedge J_7$	$P_1 \rightarrow J_3$ $P_2 \rightarrow J_8$ $P_3 \rightarrow J_4$ $P_5 \rightarrow J_1 \wedge J_2 \wedge J_5 \wedge J_6 \wedge J_7$	$P_2 \rightarrow J_1 \wedge J_3 \wedge J_4 \wedge J_5 \wedge J_6 \wedge J_7 \wedge J_8 \wedge J_{10}$ $P_6 \rightarrow J_2 \wedge J_9$	$P_2 \rightarrow J_4 \wedge J_5 \wedge J_6 \wedge J_7$ $P_3 \rightarrow J_1 \wedge J_3 \wedge J_8$ $P_5 \rightarrow J_2 \wedge J_9 \wedge J_{10}$
	Instances→	AP-3	AP-3	AP-4	AP-4
	k	4	5	2	3
	Worst solution	63	61	103	83
	Optimal solution	63	61	90	73
	Best runtime (in Sec.)	0.030	0.027	0.050	0.039
	Optimal solution corresponding schedule & alternate schedule(@)	$P_1 \rightarrow J_2$ $P_2 \rightarrow J_4 \wedge J_5 \wedge J_6 \wedge J_7$ $P_3 \rightarrow J_1 \wedge J_3 \wedge J_8$ $P_6 \rightarrow J_9 \wedge J_{10}$	$P_1 \rightarrow J_2$ $P_2 \rightarrow J_4 \wedge J_5 \wedge J_6 \wedge J_7$ $P_3 \rightarrow J_1 \wedge J_3$ $P_5 \rightarrow J_8$ $P_6 \rightarrow J_9 \wedge J_{10}$	$P_4 \rightarrow J_1 \wedge J_3 \wedge J_5 \wedge J_6 \wedge J_7 \wedge J_8$ $P_7 \rightarrow J_2 \wedge J_4 \wedge J_9 \wedge J_{10}$	$P_4 \rightarrow J_3 \wedge J_6 \wedge J_7 \wedge J_8$ $P_5 \rightarrow J_1 \wedge J_5 \wedge J_9$ $P_7 \rightarrow J_2 \wedge J_4 \wedge J_{10}$
	Instances→	AP-4	AP-4	AP-4	
	k	4	5	6	
	Worst solution	78	73	68	
	Optimal solution	68	68	68	
	Best runtime (in Sec.)	0.035	0.036	0.033	
	Optimal solution corresponding schedule & alternate schedule(@)	$P_4 \rightarrow J_3 \wedge J_6 \wedge J_7 \wedge J_8$ $P_5 \rightarrow J_9$ $P_6 \rightarrow J_1 \wedge J_5$ $P_7 \rightarrow J_2 \wedge J_4 \wedge J_{10}$	$P_2 \rightarrow J_6 \wedge J_7$ $P_4 \rightarrow J_3 \wedge J_8$ $P_5 \rightarrow J_9$ $P_6 \rightarrow J_1 \wedge J_5$ $P_7 \rightarrow J_2 \wedge J_4 \wedge J_{10}$	$P_1 \rightarrow J_4$ $P_2 \rightarrow J_6 \wedge J_7$ $P_4 \rightarrow J_3 \wedge J_8$ $P_5 \rightarrow J_9$ $P_6 \rightarrow J_1 \wedge J_5$ $P_7 \rightarrow J_2 \wedge J_{10}$	

6. Conclusion

Due to the inadequacy of the workforce, various assignment models can be seen as problems in which the number of persons is fewer than the number of jobs (i.e. $m<n$)). Thus, it is inevitable to assign multiple jobs to a single person in an effective manner. To address one such scenario, the more generalized variant of UAP called "*k*-cardinality unbalanced assignment problem" (*k-UAP*) is considered in this study. In addition, this model also includes an additional constraint over the total jobs that a person is permitted to do and a cardinality over the set of persons. This model has been formulated using 0-1 ILP. An exact LSA is developed to solve *k-UAP* optimally. Comparative results showed that the proposed LSA is competent over existing algorithm. Computational results of four test instances with distinct cardinalities have been provided. The overall results showed that the proposed LSA may be competent over the existing algorithms in providing best results of *k-UAP*.

References

Arora, S., & Puri, M. C. (1998). A variant of time minimizing assignment problem. *European Journal of Operational Research*, *110*(2), 314-325.

Bai, G. Z. (2009, December). A new algorithm for k-cardinality assignment problem. In *2009 International Conference on Computational Intelligence and Software Engineering* (pp. 1-4). IEEE.

Belik, I., & Jörnsten, K. (2016). A new Semi-Lagrangean Relaxation for the k-cardinality assignment problem. *Journal of Information and Optimization Sciences*, *37*(1), 75-100.

Bhavani, V., & Murthy, M. S. (2006). Truncated M-travelling salesmen problem. *Opsearch*, *43*(2), 152-177.

Bhunia, A. K., Biswas, A., & Samanta, S. S. (2017). A genetic algorithm-based approach for unbalanced assignment problem in interval environment. *International Journal of Logistics Systems and Management*, *27*(1), 62-77.

Dell'Amico, M., & Martello, S. (1997). The k-cardinality assignment problem. *Discrete Applied Mathematics*, *76*(1-3), 103-121.

Dell'Amico, M., Lodi, A., & Martello, S. (2001). Efficient algorithms and codes for k-cardinality assignment problems. *Discrete Applied Mathematics*, *110*(1), 25-40.

Feng, Y., & Yang, L. (2006). A two-objective fuzzy k-cardinality assignment problem. *Journal of Computational and Applied Mathematics*, *197*(1), 233-244.

Gabrovšek, B., Novak, T., Povh, J., Rupnik Poklukar, D., & Žerovnik, J. (2020). Multiple Hungarian Method for k-Assignment Problem. *Mathematics*, *8*(11), 2050.

Iampang, A., Boonjing, V., & Chanvarasuth, P. (2010, December). A cost and space efficient method for unbalanced assignment problems. In *2010 IEEE International Conference on Industrial Engineering and Engineering Management* (pp. 985-988). IEEE.

Votaw, D.F., & Orden, A. (1952) ‘The personnel assignment problem’, Symposium on Linear Inequalities and Programming, Scoop 10, US Air Force, pp.155‐163.

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, *2*(1-2), 83-97.

Kumar, A. (2006). A modified method for solving the unbalanced assignment problems. *Applied Mathematics and Computation*, *176*(1), 76-82.

Kumar, T. J., & Purusotham, S. (2017). An exact algorithm for k-cardinality degree constrained clustered minimum spanning tree problem. In *IOP Conf. Ser. Mater. Sci. Eng.* (Vol. 263, p. 042112).

Kumar, T., & Purusotham, S. (2018). The degree constrained k-cardinality minimum spanning tree problem: a lexi-search algorithm. *Decision Science Letters*, *7*(3), 301-310.

Majumdar, J., & Bhunia, A. K. (2012). An alternative approach for unbalanced assignment problem via genetic algorithm. *Applied Mathematics and Computation*, *218*(12), 6934-6941.

Malhotra, R., & Bhatia, H. L. (1984). Variants of the time minimization assignment problem. *Trab. Estad. Invest. Oper.*, *35*(3), 331-338.

Sundara Murthy, M. (1976). A bulk transportation problem. *Opsearch*, *13*(3-4), 143-155.

Thenepalle, J. K., & Singamsetty, P. (2018). Bi-criteria travelling salesman subtour problem with time threshold. *The European Physical Journal Plus*, *133*(3), 1-15.

Thenepalle, J. K., & Singamsetty, P. (2019). Lexi-search algorithm for one to many multidimensional bi-criteria unbalanced assignment problem. *International Journal of Bio-Inspired Computation*, *14*(3), 151-170.

Volgenant, A. (2004). Solving the k-cardinality assignment problem by transformation. *European Journal of Operational Research*, *157*(2), 322-331.

Yadaiah, V., & Haragopal, V. V. (2016). A new approach of solving single objective unbalanced assignment problem. *American Journal of Operations Research*, *6*(1), 81-89.