Theoretical Computer Science 618 (2016) 30–41

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Solving the Many to Many assignment problem by improving the Kuhn–Munkres algorithm with backtracking

Haibin Zhu [b], Dongning Liu [a,*], Siqin Zhang [a], Yu Zhu [c], Luyao Teng [d], Shaohua Teng [a]

[a] School of Computer Science and Technology, Guangdong University of Technology, Guangzhou, China
[b] Department of Computer Science and Mathematics, Nipissing University, North Bay, Canada
[c] Faculty of Mathematics, University of Waterloo, Waterloo, Canada
[d] College of Engineering and Science, Victoria University, Melbourne, Australia

ARTICLE INFO

Article history:
Received 23 September 2015
Received in revised form 19 November 2015
Accepted 4 January 2016
Available online 12 January 2016
Communicated by D.-Z. Du

Keywords:
Assignment problem
Many to Many assignment
Kuhn–Munkres algorithm
Hungarian method

ABSTRACT

The Many to Many (M–M) assignment problem is an important open problem where one task is assigned to many, but different, agents and one agent may undertake many, but different, tasks. The Kuhn–Munkres (K–M) algorithm is a famous and traditional process used in dealing with assignment problems. In this paper, we propose a solution to the M–M assignment problem by improving the K–M algorithm with backtracking (KM_B). To demonstrate the solution's suitability, we prove that the proposed KM_B algorithm is valid and that the worst time complexity of the KM_B algorithm is $O((\sum L^a[i])^3)$, where $L^a[i]$ denotes the maximum number of tasks that can be assigned to agent i. After that, we discuss several critical problems related to the algorithm and provide the necessary and sufficient conditions of solving the M–M assignment problem. Finally, we demonstrate, through experimentation, the validity, practicality and efficiency of the KM_B algorithm.

1. Introduction

The assignment problem is one of the fundamental combinatorial optimization problems in the branch of optimization or operations research of mathematics. The assignment problem considers that there are agents available to undertake certain tasks. Any agent can be assigned to perform any task based on agents' abilities on tasks. In collaboration, agent-task assignment aims at the highest performance based on the agents' performances [1]. Such a problem is similar to finding a maximum weight matching (or minimum weight perfect matching) in a weighted bipartite graph. A traditionally used tool is called the Hungarian method (also known as the Kuhn–Munkres algorithm) [2,3].

The Kuhn–Munkres (K–M) algorithm always requires that a task be assigned to exactly one agent who undertakes exactly one task. This constraint limits the KM algorithm to solving the One to One (1–1) assignment problem. With respect to the KM algorithm, the M–M assignment problem remains open [4].

In contrast to 1–1 assignment, the Many to Many (M–M) assignment problem allows one task to be undertaken by many, but different, agents and allows one agent to perform many, but different, tasks. For example, in a team, different people may be assigned to one or more tasks without repetition. In services, different servers may provide different services. Note that, if tasks and agents can be assigned repeatedly, it is not novel within the processing scope of the K–M Algorithm.

* Corresponding author.
E-mail address: liudn@gdut.edu.cn (D. Liu).

http://dx.doi.org/10.1016/j.tcs.2016.01.002
0304-3975/

The M–M assignment problem is common and important. In this paper, we improve the KM algorithm to solve the M–M assignment problem by introducing backtrack processes. The improved version is called the Kuhn–Munkres algorithm with Backtracking (simplified as the KM_B algorithm). To demonstrate that the proposed algorithm is correct and efficient, we prove that the KM_B algorithm is valid and the worst time complexity of the KM_B algorithm is $O((\sum L^a[i])^3)$, where $L^a[i]$ denotes the maximum number of tasks that can be assigned to agent i. After that, we discuss several critical problems with the KM_B algorithm and provide the conditions necessary, and sufficient, for the solution of the M–M assignment problem. Finally, we illustrate the validity and efficiency of the KM_B algorithm through simulations, which also verify the algorithm's practicality.

2. Literature review and preliminaries

The One to Many (1–M) assignment (Called Group Role Assignment (GRA) in [1]) is itself a complex problem where the exhaustion algorithm has an exponential increase in complexity. In this field, an efficient algorithm was developed by using the Hungarian algorithm (also called Kuhn–Munkres algorithm [2,3]). The influence of the K–M algorithm on combinatorial optimization is fundamental, where it became the prototype of a great number of algorithms in the areas of network flows, matroids, and the matching theory [5].

There are many revised and enhanced versions of the KM algorithm. For example, Bourgerois and Lassalle [6] develop an extension of this algorithm that permits a solution for rectangular matrices. Toroslu and Ücoluk [7] introduce the incremental assignment problem and propose an $O(|V|^2)$ algorithm for the problem, where $|V|$ is the size of a partition in a bipartite graph. Bertsekas and Castañon [8] discuss the parallel implementation of the auction algorithm for the classical assignment problem. Brogan [9] enhances the algorithm for ranked assignments with applications to multi-object tracking. The *k-cardinality assignment problem*, introduced by Dell'Amico and Martello [10,11], asks one to assign exactly k rows to k different columns so that the sum of the corresponding costs is the minimum. Linear bottleneck assignment problems were introduced by Fulkerson, Glicksberg and Gross [10,12] and occur, e.g., in connection with assigning jobs to parallel machines, so as to minimize the latest completion time. Grygiel [10,13] investigates the Sum-k assignment problem within an algebraic framework and designs an $O(n^5)$ algorithm in the case of real cost coefficients. Our previous work [1] extends the KM algorithm to solve the GRA (or 1–M assignment) problem. To the author's knowledge, there is a lack of fundamental and comprehensive research on the M–M assignment problem.

In engineering applications, related research work focuses on role assignment for agents in multi-agent systems [14–16] or nodes in networked systems [17]. Dastani et al. [14] present research on the determination of conditions under which an agent can adopt a role and what it means for the agent to perform a role. Durfee et al. [18] propose a new formulation of the team formation by modelling the assignment and scheduling of expert teams as a hybrid scheduling problem. Shen et al. [19] propose a multi-criteria assessment model capable of evaluating the suitability of individual workers for a specified task according to their capabilities, social relationships, and existing tasks. Stone and Veloso [20] introduce periodic team synchronization (PTS) domains as time-critical environments in which agents act autonomously. Vail and Veloso [21] extend Stone and Veloso's work [20] in role assignment and coordination in multi-robot systems, especially in highly dynamic tasks. Choi and Bahk [22] use the KM algorithm to develop a heuristic limited-matching scheduling algorithm with the partial channel feedback in OFDMA (orthogonal frequency division multiple access) and MIMO (multiple input multiple output) systems. Emms, Wilson and Hancock [23] apply the KM algorithm to recover a correspondence mapping between the graphs. Huang, Xu and Cham [24] formulate an integer programming problem to compute the distances from one probe image to all the gallery images belonging to a certain class, in which any feature of the probe image can be matched to only one feature from one of the gallery images. It is obviously that almost all of these researches are limited by instances of the 1–1 assignment problem.

Although the research above indicates a strong need to fundamentally investigate the M–M assignment problem, it is still an open problem with respect to the traditional K–M algorithm.

2.1. Problem description

To aid in the description of the M–M assignment problem, we borrow some symbols from the Environments-Classes, Agents, Roles, Groups and Objects (E-CARGO) model as follows [1], where we assume that there are n tasks and m agents in the discussions:

Definition 1 *(Task range vector).* A *task range vector* L is a vector of n tasks, where $L[j]$ denotes that quantity of task j must be assigned ($0 \le j < n$).

Definition 2 *(Ability limit vector of agent).* An *ability limit vector* of m agents is L^a, where $L^a[i]$ denotes that how many tasks can be assigned to agent i at most ($0 \le i < m$).

Definition 3 *(Performance matrix).* A performance *matrix* Q is an $m \times n$ matrix, where $Q[i, j] \in [0, 1]$ expresses the performance value of agent i ($0 \le i < m$) for task j ($0 \le j < n$). $Q[i, j] = 0$ indicates the lowest value and 1 the highest.

$$\begin{bmatrix} 0.18 & 0.82 & 0.29 & 0.01 \\ 0.35 & 0.80 & 0.58 & 0.35 \\ 0.84 & 0.85 & 0.86 & 0.36 \\ 0.96 & 0.51 & 0.45 & 0.64 \\ 0.22 & 0.33 & 0.68 & 0.33 \\ 0.96 & 0.50 & 0.10 & 0.73 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

(a) (b)

Fig. 1. A performance matrix Q and an allocation matrix T.

Definition 4 *(Allocation matrix).* An *allocation matrix* T is defined as an $m \times n$ matrix, where $T[i, j] \in \{0, 1\}$ $(0 \le i < m, 0 \le j < n)$ indicates whether agent i is assigned to task j or not. $T[i, j] = 1$ means yes and 0 no.

The M–M assignment problem allows agents to perform many but different tasks and tasks to be assigned to many but different agents. We can describe it as follows:

$$\max\left\{\sigma = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} Q[i, j] \times T[i, j]\right\}$$

subject to

$$T[i, j] \in \{0, 1\} \quad (0 \le i < m, 0 \le j < n) \tag{1}$$

$$\sum_{i=0}^{m-1} T[i, j] = L[j] \quad (0 \le j < n) \tag{2}$$

$$\sum_{j=0}^{n-1} T[i, j] \le L^a[i] \quad (0 \le i < m) \tag{3}$$

Where, expression $\max\{\sigma\}$ is the objective function of optimization; (1) is a 0–1 constraint; (2) makes the tasks be assigned to a specific number of agents, (3) makes an agent be assigned with a limited number of tasks.

For example, Fig. 1(b) is a T following the constraints $L = [1, 2, 4, 2]$ and $L^a = [1, 2, 3, 2, 2, 2]$. The sum of the assigned evaluation values is 6.57.

In fact, the M–M assignment problem is an extension of our previously discussed Group Role Assignment problem that was solved by adopting the KM algorithm [4]. Evidently, the original KM algorithm cannot be directly applied to solving this problem.

In this paper, we try to modify the KM algorithm to solve the M–M assignment problem. Therefore, we propose the K–M algorithm with backtracking (KM_B), and its worst time complexity remains cubic (i.e., $O((\sum L^a[i])^3)$). The KM_B algorithm is the major and important contribution of this paper.

In order to illustrate our improvement, we introduce a bipartite graph and a K–M theorem [10] as preliminaries for the next section (those familiar with this material may choose to skip it), because the KM algorithm aims at determining a maximum weight matching (or minimum weight perfect matching) in a weighted bipartite graph.

2.2. Bipartite graph and K–M theorem [10]

Definition 5 *(Bipartite graph).* A graph $G = (V, E)$ is *bipartite* if there exists partition $V = X \cup Y$ with $X \cap Y = \emptyset$ and $E \subseteq X \times Y$.

Definition 6 *(Matching).* A *Matching* is a subset $M \subseteq E$ such that $\forall v \in V$ at most one edge in M is incident upon v.

Definition 7 *(Perfect matching).* A *Perfect Matching* is an M in which every vertex is adjacent to some edges in M.

Definition 8 *(Weighted bipartite graph). A weighted bipartite graph* is a graph in which each edge (i, j) has a weight, or value, $w(i, j)$. The *weight* of matching M is the sum of the weights of the edges in M, $w(M) = \sum_{e \in M} w(e)$.

Definition 9 *(Complete weighted bipartite graph).* A weighted bipartite graph $G = (V, E)$ is *complete* if there exists partition $V = X \cup Y$ with $X \cap Y = \emptyset$ and $E = X \times Y$.

Let G be a complete weighted bipartite graph. The assignment problem is to find a maximum weighted matching in G. Obviously, a maximum weighted matching is perfect (cf. Definition 7).

Definition 10 *(Feasible labeling).* A vertex *labeling* is a function $\ell : V \to R$. A *feasible labeling* is one such that $\ell(x) + \ell(y) \ge w(x, y)$, $\forall x \in X$, $y \in Y$.

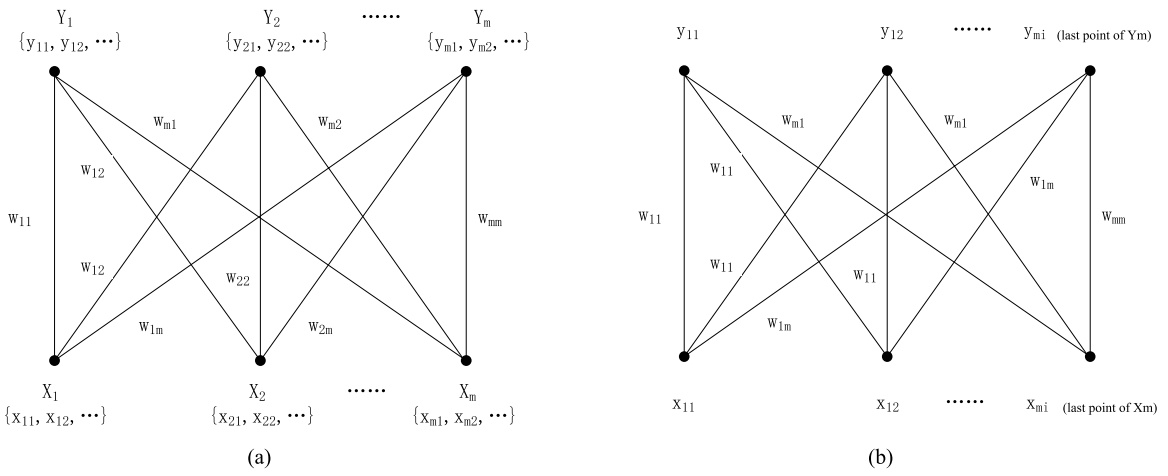


Fig. 2. (a) Bipartite graph of self-conflict. (b) Transformation of bipartite graph of self-conflict.

Definition 11 *(Equality graph).* The *Equality Graph* (with respect to ℓ) is $G = (V, E_\ell)$ where $E_\ell = \{(x, y) : \ell(x) + \ell(y) = w(x, y)\}$

Theorem 1. *(Kuhn–Munkres theorem. See [10].) If ℓ is feasible and M is a perfect matching in E_ℓ then M is a maximum weighted matching.*

3. From bipartite graph to the M–M assignment problem

3.1. Knowledge representation in bipartite graph

The K–M theorem says that for any matching M and any feasible labeling ℓ we have

$$w(M) \leq \sum_{v \in V} l(v)$$

The K–M theorem transforms the problem from an optimization problem of finding a maximum weighted matching into a combinatorial one of finding a perfect matching. It combines the weights and is a classic technique in combinatorial optimization [10].

Based on the K–M theorem, we can solve the 1–1 and 1–M assignment problems by the KM algorithm [1–3]. With respect to the M–M assignment problem, we need more transformations.

At first, the bipartite graph has to change to $G = (V, E)$, which includes partition $V = X \cup Y$ with $X \cap Y = \emptyset$ and $E \subseteq X \times Y$. Note that, X and Y are not vertex sets but sets of vertex sets, which mean that elements in X are sets, such as $X_1 = \{x_{11}, \ldots, x_{1k}\}, \ldots, X_m = \{x_{m1}, \ldots, x_{mh}\}$ $(k, h, m \geq 1)$, where x_{ij} is a vertex (It is a similar case as regards to Y). Therefore, the M–M assignment problem is becoming a second-order problem but not a first-order one.

In order to cut down the complexity of the problem, we can expand it to a first order hierarchy shown as Fig. 2(b). Then, we can deal with it by the KM_B algorithm (see in Section 4). In this section, we need to prove that if KM_B finds a perfect matching, then the matching is optimal.

We can easily unfold the bipartite graph of an M–M assignment as Fig. 2(b). Now, the problem is transformed to finding a maximum weighted matching in Fig. 2(b). According to the K–M theorem, the M–M assignment problem is to find a perfect matching M of E_ℓ by giving a feasible labeling ℓ. However, the problem is different from the traditional one in that:

1) X and Y are sets of vertex sets; and
2) There is a new constraint called *self-conflict* avoidance, i.e., if there is an edge such as $\ell_\alpha = e(x_{ij}, y_{kh})$ in one matching, then there are no other edges from set X_i to Y_k. This constraint forbids assigning repetition tasks to one agent according to the M–M assignment problem.

3.2. Optimal theorem from bipartite graph

Considering the M–M assignment with respect to the K–M algorithm, when we are starring or priming zeros, we must adjust the related elements (i.e., self-conflict elements) to be unavailable (see in Section 4). In bipartite graph G of this problem, we actually adjust the weights of edges to 0 (that are equal to ℓ_α in fact) and adjust $\ell(x)$ by $l(x) = \max_{y \in Y}\{w(x, y)\}$. After that, we can get a new bipartite graph G'.

Now, if we can find a perfect matching including ℓ_α, then this matching is the maximum weighted matching of G' according to the K–M algorithm. We need to prove that G' is the maximum weighted matching of G under the constraint of self-conflict avoidance according to the M–M assignment problem.

Theorem 2 *(The optimal theorem). If one can find a perfect matching PM in G' that includes ℓ_α, then this matching is the maximum weighted matching of G that satisfies the self-conflict avoidance constraint.*

Proof. Suppose otherwise, there exists another perfect matching PM' of G that satisfies the self-conflict avoidance constraint, such that, $w(PM) < w(PM')$. Then there exist three cases of PM' as follows:

Case 1. There is no edge that is from X_i to Y_k in PM'. In this case, it is actually equal to finding a perfect matching PM' in G' that does not include ℓ_α. By the K–M algorithm, $w(PM) = w(PM')$, i.e., this assumption is invalid.

Case 2. ℓ_α is in PM'. In this case, based on the constraint of self-conflict avoidance, there are no other edges from X_i to Y_k in PM'. It is actually equal to finding a perfect matching PM' including ℓ_α in G'. It is obviously $w(PM) = w(PM')$, then this assumption is invalid.

Case 3. ℓ_α is not in PM', but there is only one edge (assumed as ℓ_β) that is from X_i to Y_k in PM'. In this case, it is actually equal to finding a perfect matching PM' including ℓ_β in G'' (G'' is transformed from G and the weight of edges that are conflict with ℓ_β is set to zero). Because $w(\beta) = w(\alpha)$, it is obviously that $G' = G''$ and $w(PM) = w(PM')$, then this assumption is invalid, too.

Based upon the above cases, the assumption that there exists another perfect matching PM' of G that satisfies the self-conflict constraint, such that $w(PM) < w(PM')$ is invalid. The optimal theorem is proved. □

Now, the problem is transformed to whether we can find a perfect matching including ℓ_α in G'. This is why we introduce backtracking into the K–M algorithm. If we cannot find a perfect matching including ℓ_α, then we can backtrack and change the unavailable elements to available ones according to the erased starred or primed zeros in the matrix. By finite times of backtracking (less than $\sum L^a[i]$ in $\mathrm{KM_B}$), we can find the required match. To illustrate these, we introduce the $\mathrm{KM_B}$ algorithm and discuss its complexity in following sections.

4. The $\mathrm{KM_B}$ algorithm and its complexity

4.1. The K–M algorithm with backtracking (KM_B) algorithm

To solve the M–M assignment problem, we propose a new algorithm called the K–M algorithm with Backtracking (simplified as $\mathrm{KM_B}$). In the proposed $\mathrm{KM_B}$ algorithm, our goal is to avoid the situations that make $T[i, j] > 1$ $(0 \le i < m, 0 \le j < n)$. The algorithm is described as follows based on the original K–M algorithm [3], where the revised parts are underlined.

KM_B description:

- *Preparation Stage:*
  - Step 1: Create an $m \times n$ matrix called the performance matrix Q in which each element represents the performance of one of m agents doing one of n tasks.
  - Step 2: Create an ability limit vector L^a, where $L^a[i]$ tells how many tasks can be assigned to agent i at most $(0 \le i < m)$.
  - Step 3: Cardinality constraint detection, e.g., $\sum_{i=0}^{m-1} L^a[i] \ge \sum_{j=0}^{n-1} L[j]$.
  - Step 4: Expanding an $m \times n$ Q into an $K \times K$ matrix M according to vectors L and L^a, where agent i has $L^a[i]$ rows in M and task j has $L[j]$ columns in M, and $K = \sum_{i=0}^{m-1} L^a[i]$.
  - Step 5: Fill zeros: fill zeros into new columns of M such that M is a $K \times K$ matrix.
- *Processing Stage:*
  - Step 1: Reduce matrix M: for each row of M, find the smallest element and subtract it from every element in its row; after that, find the smallest element and subtract it from every element in its column.
  - Step 2: Initial Stars: find a zero (Z) in M. If there is no starred zero in its row or column, star Z, and adjust zeros to be unavailable, which belong to the same agent but in different rows or columns.
  - Step 3: Cover each column containing a starred zero. If all the columns are covered, go to Step 7; else go to Step 4.
  - Step 4: Prime some uncovered zeros: find an uncovered zero and prime it, and adjust zeros to be unavailable, which belong to the same agent but in different rows or columns (starred zeros are excluded).
    - If there is no starred zero in the row containing this primed zero, go to Step 5; else, cover this row and uncover the column containing the starred zero.
    - Continue until there are no uncovered zeros left.

- Save the smallest uncovered value and go to Step 6.

Step 5: Construct a series of alternating primed and starred zeros as follows.
- Let Z_0 represent the uncovered primed zero found in Step 4.
- Let Z_1 denote the starred zero in the column of Z_0 (if any).
- Let Z_2 denote the primed zero in the row of Z_1 (there will always be one).
- Continue until the series terminates at a primed zero that has no starred zero in its column.
- Unstar each starred zero of the series, star each primed zero of the series, erase all primes, and uncover ever line in M.
- Backtracking: adjust unavailable elements to be available according to the erased starred and primed zeros in M.
- Return to Step 3.

Step 6: Add the value found in Step 4 to every element of each covered row, and subtract it from every element of each uncovered column. Return to Step 4 without altering any stars, primes, or covered lines.

Step 7: The required assignments are indicated by the positions of the starred zeros in M. If $M[i, j]$ is a starred zero, then the element associated with row i is assigned to the element associated with column j. (End of KM_B)

The version of the K–M algorithm we revise here is put forward by Munkres in 1957 [3]. It is necessary to illustrate that in Step 2 of the *Processing Stage*, when we star zero Z, we must adjust zeros to be unavailable, which belong to the same agent but in different rows or columns. This is to forbid assigning repetitive tasks to one agent according to the new constraints of the M–M assignment problem. Zeros in the same rows or columns are not revised, because they have chances to replace Z in the future according to the Munkres's algorithm. It is the same as that for the primed zero in Step 4 of the *Processing Stage*. If we cannot find a perfect matching including Z, then we must backtrack and adjust unavailable elements to be available according to the erased starred and primed zero in Step 5 of the *Processing Stage*. We will prove that such kind of backtracking is finite in the complexity analysis, i.e., less than $\sum L^a[i]$.

The processing stage of the KM_B algorithm is as follows, where the revised parts are underlined.

Processing Stage:

Input:

∘ *A $K \times K$ matrix M with the above preliminaries after Preparation Stage.*

Output:

∘ *All the subscripts of the selected elements of M, i.e., the starred zeros.*

```
L1:  KMB(M):
L2:    {    for (all zeros z in M)
L3:           If (there are no starred zeros in the row or column of z)
L4:                star z and adjust related zeros to be unavailable;
L5:         while (true) // the main loop
L6:         { cover all the columns containing a 0 *;
L7:           if (all the columns are covered)
L8:                return; // the optimal solution is starred;
L9:           while (!(all zeros are covered)) //the inner loop
L10:          {    for (all noncovered zeros z)
L11:          {    prime z and adjust related zeros to be unavailable;
L12:               if (there is a starred zero z* in this row)
L13:               { cover row of z*;
L14:                 uncover column of z*;
L15:               }
L16:               else
L17:               { highlight z;
L18:                 while (there is a starred zero z* in z's column)
L19:                 { z = z*;
L20:                   highlight z;
L21:                   z := the primed zero in z's row; // This always exists.
L22:                   highlight z;
L23:                 }// end of while
L24:                 unstar highlighted starred zeros with backtracking;
L25:                 star highlighted primed zeros and remove highlights;
L26:                 remove primes with backtracking and uncover all rows and columns;
L27:                 continue to while (the main loop);
L28:               }// end of if
L29:          }//end of for
L30:          h := smallest uncovered element in M excluding unavailable elements;
```

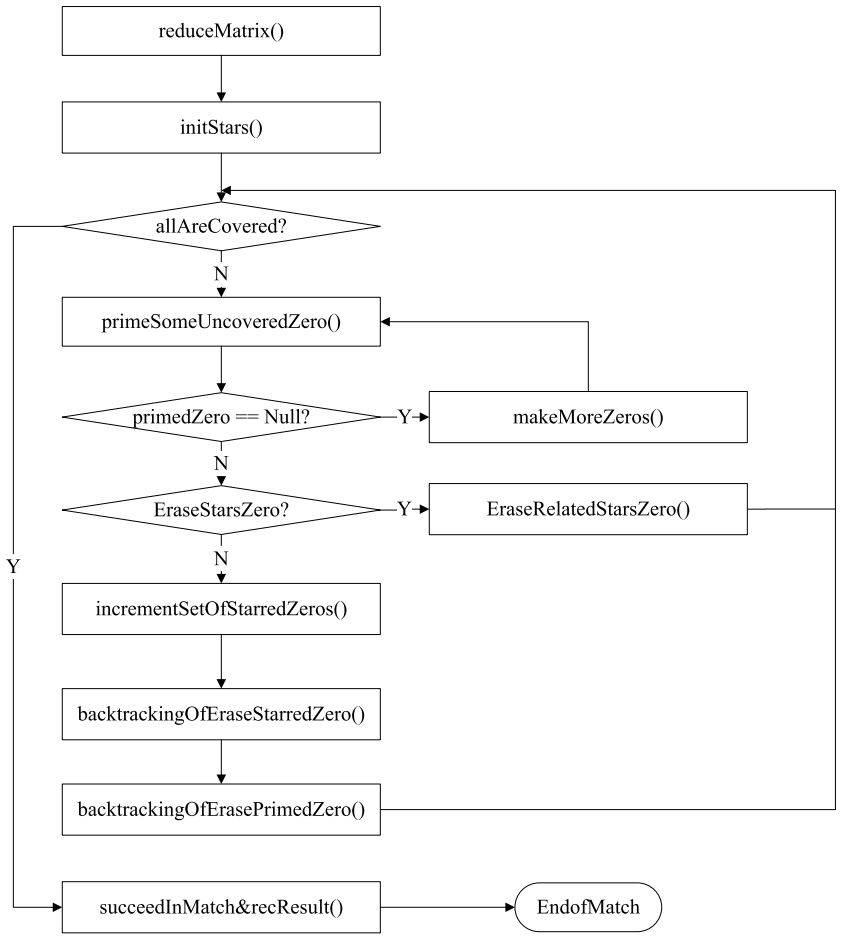


Fig. 3. Flowchart of the KM_B algorithm.

```
L31:          add h to each covered row;
L32:          subtract h from each uncovered column;
L33:        }//end of while—the inner loop
L34:      }//end of while—the main loop
L35:   }// end of KMB
```

4.2. Complexity of KM_B

To understand the complexity of the KM_B algorithm, we may use the flowchart of the KM_B algorithm (Fig. 3.):
Now, we prove that the worst time complexity of KM_B is $O((\sum L^a[i])^3)$.

Theorem 3 *(The complexity theorem). The worst time complexity of KM_B is $O((\sum L^a[i])^3)$.*

Proof. We denote that $K = \sum L^a[i]$ as the quantity of how many rows in matrix M and split KM_B as functions to illustrate.

1) Functions *reduceMatrix*(), *initStars*(), *primeSomeUncoveredZero*() and *makeMoreZeros*() are corresponding to the basic operations of Steps 1, 2, 4 and 6, whose complexity is $O(K^2)$ (the same as that in the K–M algorithm);

2) Function *incrementSetOfStarredZeros*() is corresponding to the basic operations of Step 5, whose complexity is $O(K)$ (the same as that in the K–M algorithm);

3) The complexity of cycles triggered by *EraseRelatedStarsZero*() is $O(K)$, because we need to search for a primed zero and then erase a starred zero in at most m times (while there are K zeros, the matching is found);

4) The complexity of *backtrackingOfEraseStarredZero*() and *backtrackingOfErasePrimedZero*() are determined by K, $L^a[i]$ and $L[j]$. When we backtrack an erased starred zero, we must find it in the sequence of in the increment set (the complexity is $O(K)$). After that, we need to adjust the related elements in the cycle of $L^a[i] \times L[j]$. Hence, the complexity of these functions is $O(K \times L^a[i] \times L[j])$. Because $L^a[i]$ and $L[j]$ are set to be constants and they are much less than K (an agent's

Table 1
An example of erased primed zero backtracking.

$0^{\#}$	0	0
0	b_1	b_2
$Y=0$	b_3	$X=b_4$

ability is finite, and the required number of agents to a task is finite, too), hence the run time of the latter cycle can be ignored. Thus, the complexity is $O(K)$;

5) Function *backtrackingOfErasePrimedZero*() is accompanied by function *backtrackingOfEraseStarredZero*(). The complexity of the cycles triggered by *backtrackingOfErasePrimedZero*() is $O(K)$. Note that, in function *incrementSetOfStrarredZero*(), when a starred zero is erased, two primed zeros are changed to starred zeros. Therefore, it can be backtracked no more than K times, since there are at most K starred zeros in a matching.

Based on the above, the complexity of the cycles of *allAreCovered*() is $O(K^3)$, i.e., the complexity of KM_B is $O((\sum L^a[i])^3)$. Specially, if the number of agents (m) is similar to the number of tasks (n), then $(\sum L^a[i])^3$ is in the similar orders of magnitude with respect to n. For example, if $m \geq n$, and $\sum L^a[i] = c \times m$, when c is a constant, then the complexity is $O(m^3)$. Therefore, this complexity is efficient as an algorithm within polynomial time. □

5. Some critical points in KM_B

5.1. Negative elements in backtracking

Considering the KM_B algorithm, if we cannot find a perfect matching including Z, then we must backtrack and adjust the related unavailable elements of Z to be available according to the erased starred and primed zero. In this procedure, although they are all zeros initially, these related elements may no longer be zeros, and may even become negative. If they are positive numbers or zeros, they will not affect the process of KM_B. Now, we must prove that if these related elements are negative, they will not affect KM_B either.

Theorem 4 *(The negative elements of erased primed zero). The backtracking elements of the erased primed zeros are not negative.*

Proof. Backtracking related elements of the erased primed zeros appear in the *backtrackingOfErasingPrimedZero*() function (L26 of the algorithm). For example, we can suppose the situation in Table 1.

We denote element $0^{\#}$ as the erased primed zero, and b_1, b_2, b_3, and b_4 are its related unavailable elements. If some of them are nonzeros, then these nonzeros have been processed in Step 6 of KM_B. We must prove that they are not negative:

Firstly, if $0^{\#}$ is erased, then $0^{\#}$ is not processed in the step of "Unstar each starred zero of the series, star each primed zero of the series". As a $0^{\#}$, there are no other $0^{\#}$s in its row and no other starred zeros (0^*) in its column.

Secondly, if $X = b_4$ is negative, there is no $0^{\#}$ in its row (the smallest element is not added in Step 6 of KM_B) and no 0^* in its column (the smallest element is subtracted in Step 6 of KM_B).

Thirdly, as the same column elements of $0^{\#}$, Y is certainly a zero. There is no 0^* in this column for the appearance of $0^{\#}$. Hence, Y must be subtracted by the smallest element. If Y is a zero, the optional sequence is: "$0 = Y$" → "$Y - \min + \min = 0$". Hence, there is another $0^{\#}$ in Y's row (the smallest element is added in Step 6 of KM_B).

In summary, because X and Y are in the same row, X must also go through the optional sequence: "$0 = X$" → "$X - \min + \min = 0$". Hence, X is not negative, so are b_1, b_2, b_3, etc. □

On the other side, we must prove that if the related elements of the erased starred zero are negative, they will not affect KM_B.

Theorem 5 *(The negative elements of erased starred zero). The backtracking negative elements of Erased Starred Zeros do not affect the assignment.*

Backtracking related elements of erased starred zeros appear in the *backtrackingOfErasingStarredZero*() function (L24 of the algorithm). For example, we may consider the situation in Table 2.

We denote element $X = 0^*$ as the erased starred zero, and b_1, b_2, b_3, and b_4 are its related unavailable elements. If some of them are nonzeros, then these nonzeros are not processed in Step 6 of KM_B. We denote Y and Z as zeros that are elements belonging to the same agent and task with X. Now, we have:

Case 1. If X is not substituted by Y or Z that are elements belonging to the same agent and task as that of itself in the series of "Unstar each starred zero of the series, star each primed zero of the series". In this situation, Y and Z are not $0^{\#}$, hence there are other 0^*s in Y's and Z's columns, such that b_1, b_2, b_3, and b_4 will not be assigned.

Table 2
An example of erased starred zero backtracking.

$X = 0^*$	$Y = 0^{\#}$	$Z = 0$	...	$W = 0^{\#}$
∞	b_1	b_2	...	...
∞	b_3	b_4	...	...
...	...			
$U = 0^{\#}$	...			
	$V = 0^{\#}$			

Case 2. If X is substituted by Y or Z. Without loss of generality, we assume that X is substituted by Y. Then there is a $U = 0^{\#}$, which becomes a new 0^*. We erase X in the series. If Y belongs to the final result, b_1, b_2, b_3, and b_4 will not be assigned.

After that, if Y does not belong to the final result, then there is a $V = 0^{\#}$, which becomes a new 0^*, and we erase Y in the series. There is another $W = 0^{\#}$ replacing Y. Note that, W becomes $0^{\#}$, because Y becomes 0^*. If W replaces Y, then there is certainly a $V = 0^{\#}$ becoming a new 0^* and we need to erase Y in the series. Hence, for existing V, b_1 or b_3, b_4 will not be assigned.

Especially, if Z replaces Y, then there is an iterative analysis, i.e., Y replaces X. Then, b_2 or b_4 will not be assigned either.

Based on these, the backtracking negative elements of the erased starred zeros do not affect the assignment. □

Although we prove Theorems 4 and 5, we can easily analyze that the backtracking is necessary. Because the related element may be positive or negative, if they are still zeros, then they have chances to affect the assignment and become new starred zeros. This is the reason why we must backtrack.

5.2. Necessary and sufficient conditions of processing KM_B

By looking into the M–M assignment problem, we find that there are many cases for the problem to have no feasible solutions. When we observe the performance results of the KM_B solutions, we notice that if an M–M assignment problem has no feasible solution, then it will not halt but immerse in an endless loop. Therefore, we should provide a way to check if an M–M assignment problem is decidable (has a feasible solution or not), which is the boundary of KM_B.

We prove the necessary and sufficient condition of the boundary feasible solution, which is the decidable theorem of the M–M assignment problem. To illustrate these conditions, we must give some special symbols as follows.

Given vectors U and V, we denote symbols as following:

- $|U|$ as the positive integer cardinality of *vector* U, which represents quantity of positive integer elements in U;
- $\mathrm{Max}^k U$ as the set of k biggest elements in U ($k \geq 0$), where U is a *vector* and $|U| > k$;
- U^* as the first element in U. Specially, U^* of an empty set is defined as 0;
- $\mathrm{Sub}(\mathrm{Max}^k U)$ as the function such that every k most biggest elements in U subtracts 1;
- V_0 as initial V, V_q as $V_{q-1} - \{V^*_{q-1}\}$ $(0 < q \leq \xi)$, and etc.;
- VU_0 as initial U, VU_q as $\mathrm{Sub}(\mathrm{Max}^{V^*_{q-1}} U_{q-1})$ $(0 < q \leq \xi)$, and etc.;

Theorem 6 *(Decidable theorem). The M–M assignment problem is solvable if and only if there exists a positive integer ξ, such that L_ξ is empty and for each q, $|L^*_q| \leq |L_{q-1}L^a_q|$ $(0 < q \leq \xi)$.*

Proof.

- Left to right:

Proof by contradiction: if there is not a positive integer ξ, such that L_ξ is empty and for each of q, $|L^*_q| \leq |L_{q-1}L^a_q|$ $(0 \leq q \leq \xi)$, then it is equal to that for any positive integer ξ, L_ξ is not empty or there exists an integer q, such that $L^*_q > |L_{q-1}L^a_q|$.

Case 1. (L_ξ is not empty): According to constructions of sequence of L_q and $L_{q-1}L^a_q$, it is equal to assigning tasks to agents that can fulfill the most tasks as regards to their abilities every time. This process can guarantee that $|L_{\xi-1}L^a_\xi|$ is the biggest after the current assignment, which can save the most agents for the next assignment. For any ξ, if L_ξ is not empty, there always exists at least one task that has not enough agents to fulfill it.

Case 2. ($|L^*_q| > |L_{q-1}L^a_q|$): If there exists a positive integer q, such that $|L^*_q| > |L_{q-1}L^a_q|$. It means that, in the qth time, there are not enough agents to fulfill task L^*_q.

Based on both Case 1 and Case 2, the M–M assignment problem is unsolvable.

Table 3
Test platform configuration.

	Hardware
CPU	Pentium Dual-Core CPU E5400 @2.70 GHz 2.69 GHz
MM	4 GB
	Software
OS	Windows 7 Ultimate
Eclipse	Kepler
JDK	JDK 1.7

– Right to left:

If there exists a positive integer ξ, such that L_ξ is empty and for each q, $L_q^* \leq |L_{q-1}L_q^a|$ $(0 < q \leq \xi)$, we can construct a solution by assigning tasks to agents that can undergo the largest number of tasks as regard to their abilities every time at least. The construction procedure is described in **Cardinality Constraint Detection** based on Theorem 6 described as follows. □

Cardinality Constraint Detection (CCD):
Input:
- *An* n-vector L *with the above preliminaries*; and
- *An* m-vector L^a *with the above preliminaries.*

Output:
- *True or False.*

```
L1:   CCD (L, L^a){
L2:     while (!empty(L))
L3:     { if (|L^*| > |L^a|) return false;
L4:       else {
L5:              L^a = Sub(Max^{L^*} L^a);
L6:              L = L − L^*;
L7:            }
L8:     }
L9:     return true; // true means that the problem has a feasible solution;
L10:  } end of CCD
```

We denote m as the cardinality of L^a and n as cardinality of L, which represent quantities of all elements in L^a or L. There, the complexity of the outside while loop is $O(n)$. The complexity of "$L^a = \text{Sub}(\text{Max}^{L^*} L^a)$" is $O(m \log m)$, as regards to the quick sort method in sorting L^a. Hence, the complexity of the detection algorithm is $O(n \times (m \log m))$. If m and n are similar to each other, we can abbreviate the complexity as $O(m^2 \log m)$.

6. Experiments

To check the validity and applicability of the KM_B algorithm, we conduct experiments from two perspectives. The first aspect is to compare with the exhaustion algorithm and verify the correctness of the KM_B algorithm. We randomly produce two groups of $m = 5$ and 10 (If m is bigger than 10, the processing time of the exhaustion algorithm is horrible). $L^a[i]$ belongs to [1,3] and [1,5] respectively (In reality, that a person takes more than 5 jobs is considered too much overloaded). $L[j]$ belongs to $[1, 2m/n]$ and $[1, 3m/n]$ respectively, where 2 is the mid-value of [1,3] and 3 is the mid-value of [1,5]. The performance experiments are based on the platform shown in Table 3. The performance result is shown in Table 4 and Fig. 4. The results show that the KM_B algorithm is valid.

In Fig. 4, the curves of the KM_B algorithm are in the bottoms of the figures, which are so fast that they are difficult to see. For all the successful cases, KM_B is consistent with the exhaustion algorithm. We do note that in Fig. 4(c) and Fig. 5, there are some exceptions for the efficiency of the KM_B algorithm. It is understandable that some special cases may consume more time than usual because of backtracking.

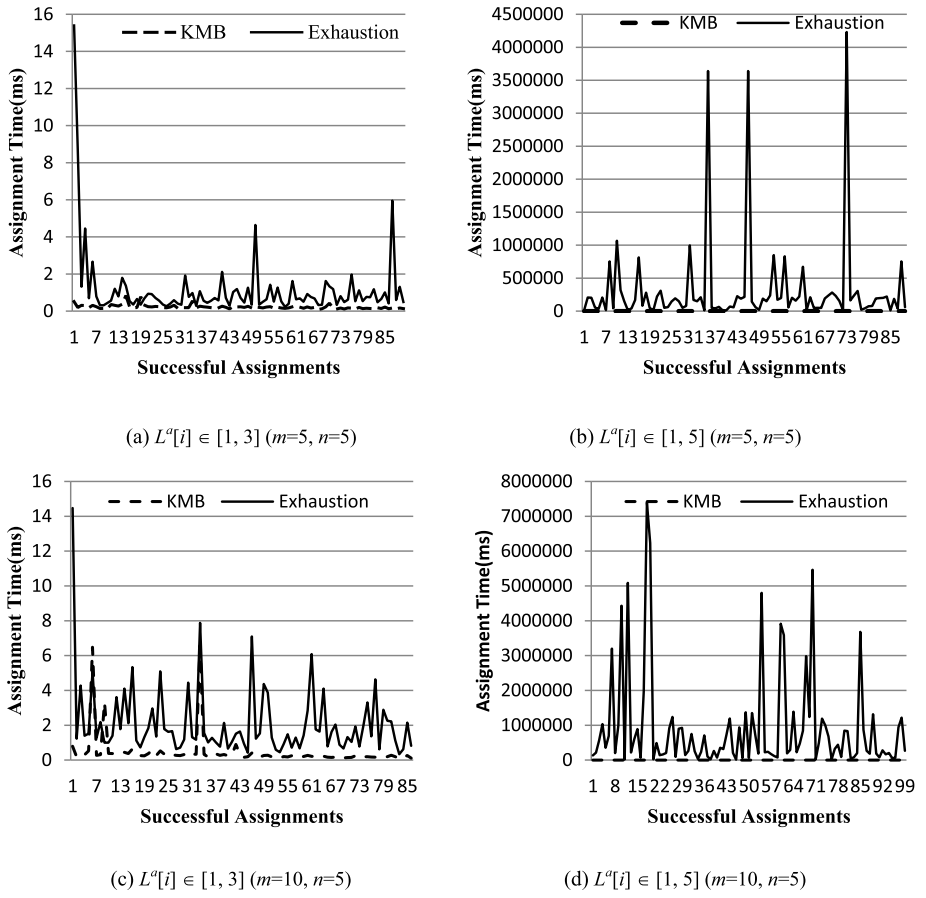
The second aspect is checking the overall performance of KM_B with random groups in different scales, where m is from 20 to 300 with a step of 20, $n = m/3$, $m/2$, m respectively, $L^a[i]$ belongs to [1,3] and [1,5] respectively, $L[j]$ is belong to $[1, 2m/n]$ and $[1, 3m/n]$ respectively, where 2 is the mid-value of [1,3] and 3 is the mid-value of [1,5]. For each $m(n)$, we produce 100 random groups and collect the average times. The performance result is shown in Fig. 5. The results show that the KM_B algorithm is efficient and practical.

7. Conclusions

The M–M assignment problem is an important open problem related to the well-known K–M algorithm. We first formalize the problem in a second order bipartite graph. Next, we solve it by improving the K–M algorithm with backtracking, i.e.,

Table 4
The experiment of comparing the KM_B with the exhaustion algorithm.

KM_B and exhaustion for 100 times ($n = 5$)								
L^a	m	Time						Successful rate
		Average (ms)		Largest (ms)		Smallest (ms)		
		KM	Exhaustion	KM	Exhaustion	KM	Exhaustion	
1–3	5	0.225	1.179	0.843	15.401	0.091	0.234	90%
1–5		0.462	2.108	6.577	14.473	0.070	0.337	86%
1–3	10	0.329	317 028	1.151	4 230 989	0.103	11 102.23	89%
1–5		0.571	929 525.8	7.902	7 398 719	0.142	11 794.33	99%



(a) $L^a[i] \in [1, 3]$ (m=5, n=5) (b) $L^a[i] \in [1, 5]$ (m=5, n=5)

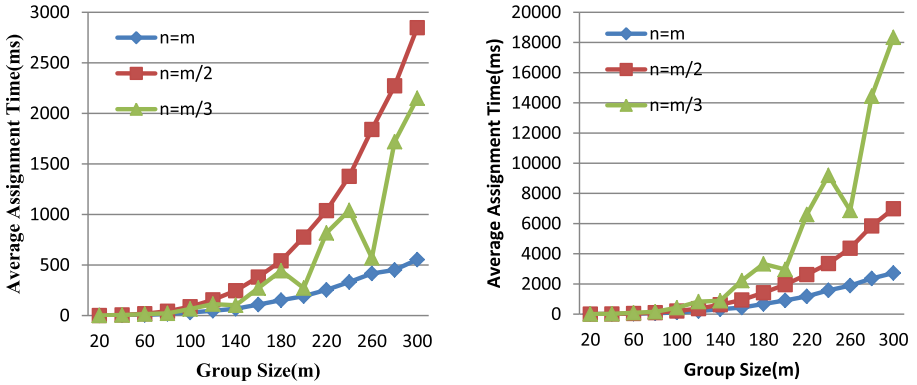
(c) $L^a[i] \in [1, 3]$ (m=10, n=5) (d) $L^a[i] \in [1, 5]$ (m=10, n=5)

Fig. 4. Four comparisons between KM_B and exhaustion.

KM_B, which is verified practical by simulation experiments. We prove that the proposed algorithm is valid and the worst time complexity of the KM_B algorithm is $O((\sum L^a[i])^3)$, where $L^a[i]$ denotes the maximum number of tasks that can be assigned to agent i. In a situation where the number of agents (m) is similar to the number of tasks (n) and $\sum L^a[i]$ is $c \times m$, where c is a constant, then $O(\sum L^a[i])^3$ is in a similar order of the magnitude with respect to m or n, i.e., $O(m^3)$.

From this paper, it is clear that further investigations may be required along the following lines:

1) Although the proposed algorithm KM_B performs on practical level, it is still possible to find more efficient implementation methods, such as parallel processing, matrix processing, skills in backtracking, space complexity optimization, and etc.
2) The M–M assignment problem may be expressed with traditional Linear Programming (LP) specifications [10,25]. It then may be solved by an optimization tool.
3) There may be additional constraints that require further investigation around the assignment problem.



(a) $L^a[i] \in [1, 3]$ (b) $L^a[i] \in [1, 5]$

Fig. 5. Two performance experiments with KM_B.

Acknowledgements

This work is supported in part by Natural Sciences and Engineering Research Council (NSERC), Canada under Grant RGPIN262075-2013 and by National Natural Science Foundation of China under Grants No. 61402118 and No. 61370229.

References

[1] H. Zhu, M.C. Zhou, R. Alkins, Group role assignment via a Kuhn–Munkres algorithm-based solution, IEEE Trans. Syst. Man Cybern., Part A, Syst. Humans 42 (3) (May 2012) 739–750.
[2] H.W. Kuhn, The Hungarian method for the assignment problem, Nav. Res. Logist. Q. 2 (1955) 83–97 (Reprinted in 52 (1) (2005) 7–21).
[3] J. Munkres, Algorithms for the assignment and transportation problems, J. Soc. Ind. Appl. Math. 5 (1) (1957) 32–38.
[4] H. Zhu, M. Zhou, M–M role-transfer problems and their solutions, IEEE Trans. Syst. Man Cybern., Part A, Syst. Humans 39 (2) (2009) 448–459.
[5] A. Frank, On Kuhn's Hungarian method – a tribute from Hungary, Naval Res. Logist. 52 (1) (2005) 2–5.
[6] F. Bourgerois, J. Lassalle, An extension of the Munkres algorithm for the assignment problem to rectangular matrices, Commun. ACM 14 (12) (1971) 802–804.
[7] I.H. Toroslu, G. Ücoluk, Incremental assignment problem, Inform. Sci. 177 (6) (2007) 1523–1529.
[8] D.P. Bertsekas, D.A. Castañon, Parallel synchronous and asynchronous implementations of the auction algorithm, Inform. Sci. 17 (6–7) (1991) 707–732.
[9] W. Brogan, Algorithm for ranked assignments with applications to multiobject tracking, J. Guid. Control Dyn. 12 (3) (1989) 357–364.
[10] R. Burkard, M. Dell'Amico, S. Martello, Assignment Problems (Revised reprint), SIAM, ISBN 978-1-61197-222-1, 2012.
[11] M. Dell'Amico, S. Martello, The k-cardinality assignment problem, Discrete Appl. Math. 76 (1997) 103–121.
[12] D.R. Fulkerson, I. Glicksberg, O. Gross, A production line assignment problem, Tech. rep. RM-1102, The Rand Corporation, Santa Monica, CA, 1953.
[13] G. Grygiel, Algebraic Σk-assignment problems, Control Cybernet. 10 (1981) 155–165.
[14] M. Dastani, V. Dignum, F. Dignum, Role-assignment in open agent societies, in: Proc. of the Second Int'l Joint Con. on Autonomous Agents and Multi-agent Systems, Melbourne, Australia, 2003, pp. 489–496.
[15] J. Ferber, O. Gutknecht, F. Michel, From agents to organizations: an organizational view of multi agent systems, in: P. Giorgini, J. Müller, J. Odell (Eds.), Agent-Oriented Software Engineering (AOSE) IV, Melbourne, in: LNCS, vol. 2935, 2004, pp. 214–230.
[16] J.J. Odell, H. Van Dyke Parunak, S. Brueckner, J. Sauter, Changing roles: dynamic role assignment, J. Object Technol. 2 (5) (September–October 2003) 77–86.
[17] M. Bhardwaj, A.P. Chandrakasan, Bounding the lifetime of sensor networks via optimal role assignments, in: Proc. of Twenty-First Annual Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM 2002), New York, USA, vol. 3, June 2002, pp. 1587–1596.
[18] E.H. Durfee, J.C. Boerkoel Jr., J. Sleight, Using hybrid scheduling for the semi-autonomous formation of expert teams, Future Gener. Comput. Syst. 31 (2014) 200–212.
[19] M. Shen, G.-H. Tzeng, D.-R. Liu, Multi-criteria task assignment in workflow management systems, in: Proc. of the 36th Annual Hawaii Int'l Conf. on System Sciences, Hawaii, USA, Jan. 6–9 2003, pp. 1–9.
[20] P. Stone, M. Veloso, Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork, Artificial Intelligence 110 (1999) 241–273.
[21] D. Vail, M. Veloso, Multi-robot dynamic role assignment and coordination through shared potential fields, in: A. Schultz, L. Parkera, F. Schneider (Eds.), Multi-Robot Systems, Kluwer, 2003, pp. 87–98.
[22] Y. Choi, S. Bahk, Multichannel wireless scheduling under limited terminal capability, IEEE Trans. Wirel. Commun. 7 (2) (2008) 611–617.
[23] D. Emms, R.C. Wilson, E.R. Hancock, Graph matching using the interference of continuous-time quantum walks, Pattern Recognit. 42 (2009) 985–1002.
[24] Y. Huang, D. Xu, T. Cham, Face and human gait recognition using image-to-class distance, IEEE Trans. Circuits Syst. Video Technol. 20 (3) (2010) 431–438.
[25] N. Karmarkar, A new polynomial-time algorithm for linear programming, Combinatorica 4 (4) (1984) 373–395.