arXiv:1012.1288v5 [cs.DC] 11 Feb 2016

Representations of task assignments in distributed systems using Young tableaux and symmetric groups

Dohan Kim*

A.I. Research Co., 2537-1 Kyungwon Plaza 201, Sinheung-dong, Sujeong-gu, Seongnam-si, Kyunggi-do, 461-811, South Korea

This paper presents a novel approach to representing task assignments for partitioned agents (respectively, tasks) in distributed systems. A partition of agents (respectively, tasks) is represented by a Young tableau, which is one of the main tools in studying symmetric groups and combinatorics. In this paper we propose a task, agent, and assignment tableau in order to represent a task assignment for partitioned agents (respectively, tasks) in a distributed system. This paper is concerned with representations of task assignments rather than finding approximate or near optimal solutions for task assignments. A Young tableau approach allows us to raise the expressiveness of partitioned agents (respectively, tasks) and their task assignments.

Keywords: Young tableau; task assignment; symmetric group; distributed agents

1. Introduction

A distributed system is defined to be a collection of independent nodes that appear as a single coherent computer [42]. Parallel agents [8, 23, 44] in a distributed system often take advantage of parallelism [11, 42] by dividing a job into many tasks that execute on one or more agents. The primary purposes of task assignments in distributed systems are to increase the system throughput and to improve resource utilization [10, 27, 36, 38, 47]. A subclass of task assignment problems involves an equal number of tasks and agents, where the mapping between a set of tasks and a set of agents is bijective. One of its fundamental form is represented by the *linear assignment problem* [4–6, 29], which concerns how n tasks are assigned to n agents in the best possible way. Meanwhile, if tasks have a precedence relationship, they can be expressed as a directed acyclic graph (DAG), where each node represents a task and each edge represents a precedence constraint [39]. We focus on the representation of n-task-n-agent assignments for a given acyclic task graph $G = (V, E)$ in a distributed system. In our approach an n-task-n-agent assignment is represented by a *Young tableau* [15]. Our approach to representing an n-task-n-agent assignment is quite general, aiming to apply for task assignments involving the same number of tasks and agents in other disciplinary areas, such as robotics [28] and operations research [4].

We also describe n-task-m-agent assignments ($n > m$) by using *generalized Young tableaux* and discuss their task reassignments by means of a group action on a set of generalized Young tableaux.

---

* Email: dkim@airesearch.kr

The remainder of this paper is organized as follows. We describe a task assignment problem including the n-task-n-agent assignment problem in a distributed system in Section 2. Section 3 provides an introduction to groups and Young tableaux. In Section 4 we discuss how an n-task-n-agent assignment can be represented by an element of a *symmetric group* $\mathfrak{S}_n$ [33]. Section 5 presents our approach to representing task assignments for agents in a distributed system using Young tableaux. We discuss an equivalence relation on a set of Young tableaux for n-task-n-agent assignments in this section. We also discuss generalized Young tableaux for n-task-m-agent assignments $(n > m)$ and their equivalence relation on a set of generalized Young tableaux in this section. In Section 6 we discuss the counting aspect of the search space involving n-task-n-agent assignments and their reassignments by using *tabloids* and symmetric groups [33]. Finally, we conclude in Section 7.

2. Task assignments in distributed systems

2.1 *Task assignment problem in a distributed system*

A task assignment problem in a distributed system is found in [10, 27, 35, 36] and is defined as follows:

Let T be a set of n tasks such that $T = \{t_1, t_2, \ldots, t_n\}$ and let A be a set of m $(m \leq n)$ agents[1] such that $A = \{a_1, a_2, \ldots, a_m\}$. Each task and agent is not necessarily homogeneous in a distributed system. A partial order relation $\prec$ can be defined on T, which specifies a task precedence constraint. For any two tasks $t_i, t_j \in T$, $t_i \prec t_j$ denotes that t_i must be completed before t_j can begin. Let M be a task assignment between T and A. Let $t_a^e(M)$ be the total execution time of agent a for the task assignment M, and $t_a^i(M)$ be the total idle time of agent a for the task assignment M. Agent a is idle before the execution of its first task or between the executions of its two consecutive tasks for the task assignment M. The turnaround time of agent a is the total time spent in agent a for the task assignment M. Let $t_a(M) := t_a^e(M) + t_a^i(M)$ and $t(M) := \max_a t_a(M)$, where $t(M)$ is called the *task turnaround time* of the task assignment M. In contrast, let $u_a(M) := t_a^e(M)/t(M)$, where $u_a(M)$ is the agent utilization of agent a for the task assignment M. The *average agent utilization* for the task assignment M is defined to be the average agent utilization for m agents, i.e., $u(M) := (\sum_{k=1}^{m} u_{a_k}(M))/m$. If the performance metric for a task assignment is the task turnaround time, an optimal task assignment is defined to be the task assignment M_0 such that

$$t(M_0) := \min_M t(M) = \min_M \max_a t_a(M) \ .$$

If the performance metric for a task assignment is the average agent utilization, an optimal task assignment is defined to be the task assignment M_0 such that

$$u(M_0) := \max_M u(M) \ .$$

Constraints and assumptions are as follows:

(1) Both tasks and distributed agents are not necessarily homogeneous and the information regarding their characteristics is available to a task assignment system. Agents are dedicated to a task assignment in which no other task or job is involved when a task assignment is executed.

---

[1]In this paper we use *agent* and *processor* interchangeably if parallel agents in a distributed system are considered as simple computing entities [44].

(2) The network of distributed agents is fully-connected in which communication links are identical with the same data transfer rate. Communications between agents take place by message passing.

(3) Each task is assigned to exactly one agent in such a way that each agent is able to process only single task at a time in a non-preemptive manner. It is also required that at least one task is assigned to each agent.

(4) Precedence constraints can be imposed. A task t_j can be executed if all its predecessors t_i with $t_i \prec t_j$ have completed. A task graph is directed and acyclic.

Traditional approaches to representing task assignments have limitations in some cases. For instance, an assignment is often represented as a set of pairs (task ID, agent ID), graphs, matrices, charts, or tables [4, 10, 36]. When we apply a logical partition to agents (or tasks) and their task assignments, those approaches often lack a systematic way of expressing a partition. Note that a partition in this paper refers to a logical partition of tasks or agents, which is different from the partition used in the context of *grain packing* [39] that concerns how to partition a job into subtasks in order to improve the performance criteria. In our approach task assignments are represented by Young tableaux or tabloids in which partitions are naturally expressed. In Section 2.2 we provide the definitions and terminology for task assignments used in this paper.

2.2 *Definitions and terminology for task assignments*

In this subsection we introduce definitions and terminology for task assignments in a heterogeneous (agent-based) system. Definitions and terminology used in this subsection are found in [2, 12, 22, 35, 39, 41, 47].

A *task graph* $T = (V, E)$ is a directed acyclic graph, where each node in $V = \{v_1, v_2, \ldots, v_n\}$ denotes a task, and each edge $(v_i, v_j) \in E \subset V \times V$ denotes a precedence relationship between tasks, i.e., v_j cannot begin before v_i completes. The positive weight associated with each task $v \in V$ represents a computation requirement. The nonnegative weight associated with each edge $(v_i, v_j) \in E$ represents a communication requirement.

A fully-connected heterogeneous system A is a set $A = \{a_1, a_2, \ldots, a_m\}$ of m heterogeneous agents whose network topology is fully-connected. A heterogeneous system A is called *consistent* if agent $a_i \in A$ executes a task n-times faster than agent $a_j \in A$, then it executes all other tasks n-times faster than agent a_j. A heterogeneous system A is called *communication homogeneous* if each communication link in A is identical.

Let $T = (V, E)$ be a task graph and $A = \{a_1, a_2, \ldots, a_m\}$ be a fully-connected heterogeneous system. Assume that a startup cost of initiating a task on an agent is negligible. In a consistent system, the computation cost of task v_i on a_j is $\omega(v_i, a_j) = r(v_i)/e(a_j)$, where $r(v_i)$ is the computation requirement of task v_i, and $e(a_j)$ is the execution rate of agent a_j. Meanwhile, in an inconsistent system, the computation cost of task v_i on a_j is given by $\omega(v_i, a_j) = w_{ij}$, where w_{ij} is the $(i, j)^{\text{th}}$ entry in a $|V| \times |A|$ cost matrix W. Note that an inconsistent system model is a generalization of a consistent system model. Now, the communication cost model of A is defined as follows. Let $d(v_i, v_j)$ be the amount of data to be transferred from task v_i to task v_j for each $(v_i, v_j) \in E$; let $t(a_s, a_t)$ be the data transfer rate between the communication link between agent a_s and a_t in A. Let M be a task assignment between T and A such that $M(v_i) = a_s$ and $M(v_j) = a_t$ for $v_i, v_j \in T$ and $a_s, a_t \in A$. Assume that both local communication and communication startup cost are negligible. If the communication of A has the linear cost model [2], the communication cost between task v_i on agent a_s and task v_j on a_t is given by $c(M(v_i), M(v_j)) = d(v_i, v_j)/t(a_s, a_t)$ if $a_s \neq a_t$, and 0 otherwise. Furthermore, if A is communication homogeneous such that the data transfer rate of each communication link

is 1, the communication requirement and the communication cost coincide for inter-agent communication.

Let $T = (V, E)$ be a task graph and $A = \{a_1, a_2, \ldots, a_m\}$ be a fully-connected heterogeneous system. Suppose task $v_i \in V$ is assigned to agent $a_j \in A$, and a start time of task v_i is $t_s(v_i, a_j)$. Then, the *finish time* of task v_i on agent a_j is

$$t_f(v_i, a_j) := t_s(v_i, a_j) + \omega(v_i, a_j).$$

Let $T = (V, E)$ be a task graph and $A = \{a_1, a_2, \ldots, a_m\}$ be a fully-connected heterogeneous system. The earliest possible start time of task $v_j \in V$ on agent $a_k \in A$ is called the *data ready time*, which is defined as

$$t_{\mathrm{dr}}(v_j, a_k) := \max_{v_i \in \mathrm{pred}(v_j)} \{t_f(v_i, M(v_i)) + c(M(v_i), M(v_j))\},$$

where $\mathrm{pred}(v_j)$ denotes the set of all predecessors of task v_j, and $M(v_i)$ denotes the agent to which task v_i is assigned by a task assignment M. If $\mathrm{pred}(v_j) = \emptyset$, then v_j is called an *entry node*, and it is assumed that $t_{\mathrm{dr}}(v_j, a_k) = 0$ for all $a_k \in A$.

2.3 The n-task-n-agent assignment problem in a distributed system

The n-task-n-agent assignment problem is a task assignment problem in a distributed system, which involves the same number of tasks and agents (cf. linear assignment problem [4–6]). In the remainder of this paper, a target heterogeneous system A for the n-task-n-agent assignment problem is assumed to be fully-connected, consistent, and communication homogeneous, where the communication requirement and the communication cost coincide. Figure 1 shows a task graph with eight tasks and examples of 8-task-8-agent assignments A_i for $1 \leq i \leq 3$. The label next to each node in the task graph denotes the computation requirement and the label next to each edge denotes the communication requirement or communication cost. For instance, the communication cost between task 1 and 2 is 5 time units.

(a)

(b)

T: Task, C: Computation requirements (τ)
A: Agent, E: Execution rate (τ/u)

A_i (i=1,2,3): Examples of the equivalent 8-task-8-agent assignments

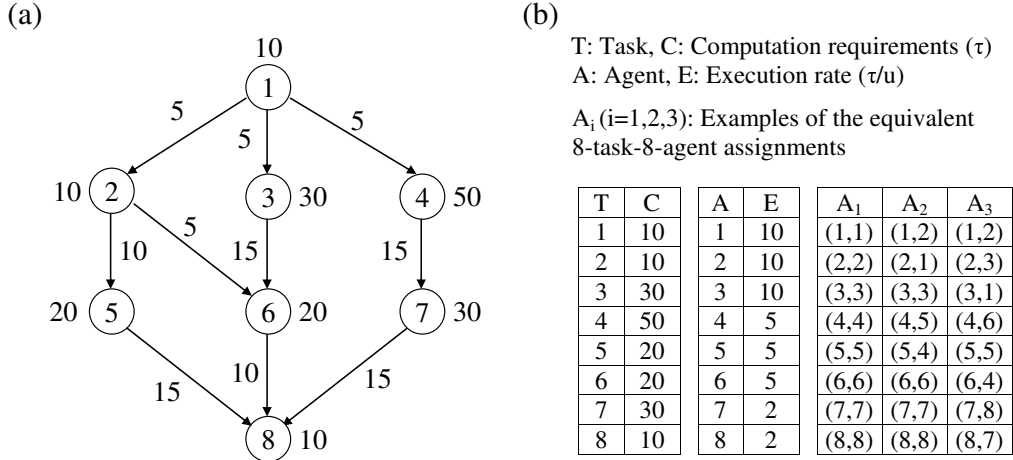| T	C		A	E		A_1	A_2	A_3
1	10		1	10		(1,1)	(1,2)	(1,2)
2	10		2	10		(2,2)	(2,1)	(2,3)
3	30		3	10		(3,3)	(3,3)	(3,1)
4	50		4	5		(4,4)	(4,5)	(4,6)
5	20		5	5		(5,5)	(5,4)	(5,5)
6	20		6	5		(6,6)	(6,6)	(6,4)
7	30		7	2		(7,7)	(7,7)	(7,8)
8	10		8	2		(8,8)	(8,8)	(8,7)

Figure 1. (a) Task graph $G = (V, E)$; (b) characteristics of tasks, agents, and examples of 8-task-8-agent assignments.

Consider the task assignment A_1 in Figure 1(b). Each (a, b) in A_k for $1 \leq k \leq 3$ denotes that task a is assigned to agent b. Task 1 is the entry node in the task graph $G = (V, E)$, so it starts at time 0. Since task 1 is assigned to agent 1 in A_1, the computation cost of task 1 is its computation requirement divided by the execution rate of agent 1. A possible choice of units for τ and u in Figure 1(b) are Flop (Floating-point operation) [43] and

second, respectively. Note that each task is assigned to each agent in such a way that their IDs are the same in A_1 (see Figure 1(b)). Therefore, the computation cost of task 1 on agent 1 is $10/10 = 1$ time unit in A_1. Since each task is assigned to a distinct agent for the n-task-n-agent assignment problem, each task starts at its data ready time. Thus, task 2 on agent 2 starts its execution at $1 + 5 = 6$ time units. Similarly, task 3 on agent 3 and task 4 on agent 4 start at 6 time units. Simple calculations show that task 5 on agent 5 starts at 17, task 6 on agent 6 at 24, task 7 on agent 7 at 31, and task 8 on agent 8 at 61 time units. Thus, the task turnaround time of A_1 is $61 + 10/2 = 66$ time units, where $10/2$ is a computation cost of task 8 on agent 8. Note that the execution rate of agents 1, 2, and 3 are the same (see Figure 1(b)). Thus, it is indistinguishable in terms of the task turnaround time if we swap agents with the same execution rate in a given n-task-n-agent assignment. We see that the task turnaround time of A_2 and A_3 are the same with that of A_1. Furthermore, once the spatial assignment of tasks (i.e., allocation of tasks to agents [39]) has been determined, the temporal assignment of tasks (i.e., attribution of a start time to each task [39]) is deterministic for the n-task-n-agent assignment problem, i.e., the start time of each task on an agent is always its unique data ready time. Note also that if tasks in the n-task-n-agent assignment problem have no precedence relationship, the start time of each task on an agent is simply 0.

Traditional methods [10, 27, 36, 47] to representing task assignments have some limitations if task assignments involve the same number of tasks and agents. If we apply an equivalence relation to tasks or agents, traditional approaches do not naturally express those assignments that belong to an equivalence class. We partition the search space $\mathfrak{S}_n$ of n-task-n-agent assignments by using an equivalence relation of Young tableaux of a given tableau shape. We introduce the necessary definitions and results of group theory and Young tableaux in Section 3.

3.   Groups and Young tableaux

Group theory is a branch of mathematics, which provides the methods, among other things, to analyze symmetry in both abstract and physical systems [48]. In this section we give definitions on groups and Young tableaux used in this paper. Definitions and results in this section are found in [1, 9, 13, 15, 16, 21, 24, 33, 50].

A *group* $(G, \cdot)$ is a nonempty set G, closed under a binary operation $\cdot$, such that the following axioms are satisfied: (i) $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in G$, (ii) there is an identity element $e \in G$ such that for all $x \in G$, $e \cdot x = x \cdot e = x$, (iii) for each element $a \in G$, there is an element $a^{-1} \in G$ such that $a \cdot a^{-1} = a^{-1} \cdot a = e$.

The *order* of a finite group G, denoted $|G|$, is the number of elements of G.

Let G be a group and H be a nonempty subset of a group G. If H itself is a group under the restriction to H of the binary operation of G, then H is a *subgroup* of G, denoted by $H \leq G$. A subset H of a group G is a subgroup of G iff (i) H is closed under the binary operation of G, (ii) the identity e of G is in H, (iii) $h^{-1} \in H$ whenever $h \in H$.

Let $I_n = \{1, 2, \ldots, n\}$. The group of all bijections $I_n \to I_n$, whose binary operation is function composition, is called the *symmetric group on n letters* and denoted $\mathfrak{S}_n$. Since $\mathfrak{S}_n$ is the group of all permutations of a set $I_n = \{1, 2, \ldots, n\}$, the order of $\mathfrak{S}_n$, i.e., $|\mathfrak{S}_n|$, is $n!$. A subgroup of a symmetric group is called a *permutation group*.

A permutation $\begin{pmatrix} 1 & 2 & \cdots & n \\ a_1 & a_2 & \cdots & a_n \end{pmatrix} \in \mathfrak{S}_n$ is written in the *two-line notation*, while $a_1 a_2 \cdots a_n \in \mathfrak{S}_n$ is written in the *one-line notation* [33].

Let $i_1, i_2, \ldots, i_r$ $(r \leq n)$ be distinct elements of $I_n = \{1, 2, \ldots, n\}$. Then $(i_1 i_2 \cdots i_r)$ is defined to be the permutation that maps $i_1 \mapsto i_2$, $i_2 \mapsto i_3$, $\ldots$, $i_{r-1} \mapsto i_r$ and $i_r \mapsto i_1$, and every other element of I_n maps onto itself. $(i_1 i_2 \cdots i_r)$ is called a cycle of length r

or an *r-cycle* [21]. A 2-*cycle* is called a *transposition* [13, 21].

For instance, permutation $p = 3\,1\,4\,2 \in \mathfrak{S}_4$ is a 4-cycle such that $(1\,3\,4\,2) = (3\,4\,2\,1) = (4\,2\,1\,3) = (2\,1\,3\,4)$.

Every permutation of a finite set can be written as a product of disjoint cycles. Any permutation of a finite set of at least of two elements can be written as a product of transpositions. If $p \in \mathfrak{S}_n$ is the product of disjoint cycles of lengths $l_1, l_2, \ldots, l_r$ with $l_1 \leq l_2 \leq \cdots \leq l_r$ (including its 1-cycles), the integers $l_1, l_2, \ldots, l_r$ are called the *cycle type* of p.

Let G be a group and let $s_i \in G$ for $i \in I$. The subgroup generated by $S = \{s_i : i \in I\}$ is the smallest subgroup of G containing the set S. If this subgroup is all of G, then S is called a *generating set* of G.

Let G be a group. For any $H \leq G$ and any $g \in G$, let $gH = \{gh : h \in H\}$ and $Hg = \{hg : h \in H\}$. The former is called the *left coset* of H in G and the latter is called the *right coset* of H in G.

Let G be a group and let $H \leq G$. A subset $T = \{t_i\}$ of G is called a *(left) transversal* for H in G if the set T consists of precisely one element from each left coset of H in G.

Let G be a group and let $H \leq G$. The number of left cosets of H in G is called the *index* $[G : H]$ of H in G. If G is a finite group, $[G : H] = |G|/|H|$.

Let G be a group whose identity element is e. A (left) *action* of G on a set X is a function $G \times X \to X$ such that for all $x \in X$ and $g_1, g_2 \in G$: (i) $ex = x$, (ii) $(g_1 g_2)x = g_1(g_2 x)$. When such an action is given, we say that G acts (left) on the set X, and X is called a G-set.

Let X be a set. A relation $\sim_R$ on $X \times X$ is called an *equivalence relation* on X provided $\sim_R$ is: (i) reflexive: $x \sim_R x$ for all $x \in X$, (ii) symmetric: $x \sim_R y \Rightarrow y \sim_R x$, (iii) transitive: $x \sim_R y$ and $y \sim_R z \Rightarrow x \sim_R z$. The *equivalence class* of $x \in X$ under $\sim_R$ is defined to be the set $\{y \in X : y \sim_R x\}$.

Let X be a G-set. For $x_1, x_2 \in X$, let $x_1 \sim_R x_2$ iff there exists $g \in G$ such that $gx_1 = x_2$. Then, $\sim_R$ is an equivalence relation on X. The equivalence classes with respect to $\sim_R$ are called the *orbits* of G on X.

A *partition* of n is defined to be a sequence $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_i)$, where the λ_j are weakly decreasing and $\sum_{j=1}^{i} \lambda_j = n$. If λ is a partition of n, then it is denoted $\lambda \vdash n$.

Let $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_i) \vdash n$. A *Young diagram* (or *Ferrers diagram*) of shape λ is a left-justified, finite collection of cells, or boxes, with row j containing λ_j cells for $1 \leq j \leq i$.

Let $\lambda \vdash n$. A *Young tableau* of shape λ is an array t, obtained by assigning numbers in $\{1, 2, \ldots, n\}$ to the cells of the *Young diagram* of shape λ bijectively.

Let $\lambda \vdash n$. A *generalized Young tableau* of shape λ is a filling of the *Young diagram* of shape λ with positive integers (repetitions allowed).

For instance, let $\lambda = (2, 1)$. The list of all Young tableau of the shape λ is as follows:

$$\begin{array}{cc}1&2\\\hline 3\end{array} \,,\quad \begin{array}{cc}1&3\\\hline 2\end{array} \,,\quad \begin{array}{cc}2&1\\\hline 3\end{array} \,,\quad \begin{array}{cc}2&3\\\hline 1\end{array} \,,\quad \begin{array}{cc}3&1\\\hline 2\end{array} \,,\quad \begin{array}{cc}3&2\\\hline 1\end{array} \,.$$

Two Young tableaux t_1, t_2 of the same shape λ are called *row equivalent*, denoted $t_1 \sim t_2$, if the corresponding rows of t_1 and t_2 contain the same elements. (The reader is encouraged to verify that $\sim$ is an equivalence relation on the set of Young tableaux of shape λ.) A *tabloid* of shape λ is an equivalence class, defined as $\{t\} = \{t_1 : t_1 \sim t\}$, where the shape of t is λ.

To denote a tabloid $\{t\}$, only horizontal lines between rows are used. For instance,

$$t = \begin{array}{cc}1&2\\\hline 3\end{array} \quad \text{implies} \quad \{t\} = \left\{ \begin{array}{cc}1&2\\\hline 3\end{array} \,, \begin{array}{cc}2&1\\\hline 3\end{array} \right\} = \begin{array}{c}\underline{1\quad 2}\\ 3\end{array} \,.$$

Lemma 3.1 ([33]). *Suppose* $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_i) \vdash n$. *For a given tabloid of shape* λ, *the number of Young tableaux of shape* λ *in the tabloid is* $\lambda_1! \lambda_2! \cdots \lambda_i!$.

Lemma 3.2 ([33]). *Let $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_i) \vdash n$. The number of distinct tabloids of shape λ is $n!/(\lambda_1!\lambda_2! \cdots \lambda_i!)$.*

Lemmas 3.1 and 3.2 can be obtained by using simple combinatorial arguments. The interested reader may refer to [33] for further details.

4. Representations of *n*-task-*n*-agent assignments using a symmetric group

Representations of bijective task assignments between tasks and agents (or processors) using a group theory have already been researched in [24, 32]. In this section we summarize how an n-task-n-agent assignment is represented by an element of $\mathfrak{S}_n$.

Let U_n be a set of n tasks and W_n be a set of n distributed agents such that $U_n = W_n = \{1, 2, \ldots, n\}$. Then the group of all bijections $U_n \to W_n$ is $\mathfrak{S}_n$, where each element of $\mathfrak{S}_n$ denotes each n-task-n-agent assignment between U_n and W_n. Therefore, a permutation $\begin{pmatrix} 1 & 2 & \cdots & n \\ a_1 & a_2 & \cdots & a_n \end{pmatrix} \stackrel{\text{def}}{=} a_1 a_2 \cdots a_n \in \mathfrak{S}_n$ [33] can be used to denote an n-task-n-agent assignment that maps $1 \mapsto a_1, 2 \mapsto a_2, \ldots, n \mapsto a_n$, where $1, 2, \ldots, n \in U_n$ and $a_1, a_2, \ldots, a_n \in W_n$ such that $U_n = W_n = \{1, 2, \ldots, n\}$.

Let $U_4 = W_4 = \{1, 2, 3, 4\}$. If permutation $p = 3\,1\,4\,2 = (1\,3\,4\,2) \in \mathfrak{S}_4$ is used to denote a n-task-n-agent assignment for $n = 4$, then it can be represented by the set $\{(1 \mapsto 3), (2 \mapsto 1), (3 \mapsto 4), (4 \mapsto 2)\}$, where $(a \mapsto b)$ denotes that task a is assigned to agent b for $a \in U_4$ and $b \in W_4$.

Although an n-task-n-agent assignment can be represented by the above manner, a task assignment for partitioned agents (or tasks) is not naturally represented. We will discuss this issue in the next section.

Meanwhile, a reassignment of an n-task-n-agent assignment can be represented by using permutation multiplication. For instance, if permutation $q = i_1 i_2 \cdots i_n \in \mathfrak{S}_n$ is used to denote an n-task-n-agent assignment, then the right multiplication of q by transposition $(i\,j) \in \mathfrak{S}_n$ may represent the swapping of the task in agent i and the task in agent j [24]. Permutation multiplication is further discussed in [13, 24].

5. Assignment tableaux and tabloids

5.1 *Assignment tableaux and tabloids for n-task-n-agent assignments*

In this subsection we present our approach to representing n-task-n-agent assignments by using Young tableaux and tabloids. We first introduce a *task tableau* and an *agent tableau*. Then, we define an *assignment tableau*, which is a 2-tuple of a task and agent tableau.

Definition 5.1. Suppose $\lambda \vdash n$. A *task tableau* of shape λ, denoted t_λ, is a Young tableau of shape $\lambda \vdash n$, obtained by assigning tasks (i.e., task IDs) in $\{1, 2, \ldots, n\}$ to the cells of the *Young diagram* of shape λ bijectively. An *agent tableau* of shape λ, denoted a_λ, is a Young tableau of shape λ, obtained by assigning agents (i.e., agent IDs) in $\{1, 2, \ldots, n\}$ to the cells of the *Young diagram* of shape λ bijectively.

Suppose fourteen agents are partitioned into $\{1, 3, 8, 14\}$, $\{2, 5, 6, 4\}$, $\{9, 7, 12\}$, and $\{10, 13, 11\}$. It is naturally represented as an agent tableau in Figure 2(a). Suppose further that the 14-task-14-agent assignment is given by $\{(1 \mapsto 1), (3 \mapsto 3), (5 \mapsto 8), (11 \mapsto 14), (2 \mapsto 2), (4 \mapsto 5), (8 \mapsto 6), (6 \mapsto 4), (7 \mapsto 9), (9 \mapsto 7), (10 \mapsto 12), (13 \mapsto 10), (12 \mapsto 13), (14 \mapsto 11)\}$, where $(a \mapsto b)$ means task a is assigned to agent b. This task assignment may have a compact form of a representation as shown in Figure 2(b), where the entry in
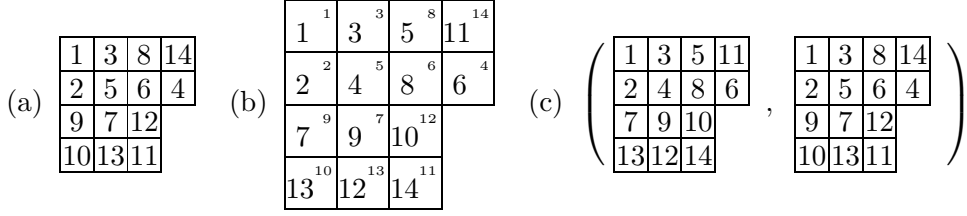
Figure 2. (a) An agent tableau; (b) a compact form of an assignment; (c) an assignment tableau.

each cell represents a task and the label in the upper right corner of each cell represents an agent. Since Figure 2(b) is not a standard form of a Young tableau, we describe this task assignment as a 2-tuple of Young tableaux instead. We use the task tableau of the same shape with that of the agent tableau as shown in Figure 2(c) in order to represent the task assignment corresponding to Figure 2(b). Now we define an *assignment tableau* to represent an n-task-n-agent assignment.

Definition 5.2. An *assignment tableau* of shape λ, denoted s_λ, is a 2-tuple of Young tableaux $s_\lambda \stackrel{\text{def}}{=} (t_\lambda, a_\lambda)$, where t_λ is a task tableau of shape λ and a_λ is an agent tableau of shape λ.

An assignment tableau s_λ represents a task assignment, where each task in a cell (i, j) of t_λ is assigned to each agent in a cell (i, j) of a_λ bijectively. Therefore, we also denote s_λ as the set of all $(a \mapsto b)$ [25], where a is a task in a cell (i, j) of t_λ and b is an agent in a cell (i, j) of a_λ.

Definition 5.3. A *standard agent tableau*[1] of shape λ, denoted A_λ, is an agent tableau having the entries of agent IDs $\{1, 2, \ldots, n\}$ in a sequential order, starting from the top left and ending at the bottom right. If an agent tableau is a standard agent tableau, then we say that the associated assignment tableau is *standard*, denoted S_λ.

Figure 3. (a) Assignment tableau; (b),(c) standard assignment tableaux.

A simple renaming of agent IDs can be applied if necessary, in order to convert from an existing agent tableau of shape $\lambda \vdash n$ to the standard agent tableau of the same shape. Note that if an agent tableau is of shape $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_j)$, then the $\lambda_i (1 \leq i \leq j)$ are weakly decreasing by the partition constraint of Young tableau. The following algorithm describes a simple renaming of agent IDs to convert from an existing agent tableau of shape $\lambda \vdash n$ to the standard agent tableau of the same shape. (It is exactly the same way to rename the task IDs in order to convert from an existing task tableau of shape $\lambda \vdash n$ to the standard task tableau of the same shape. Note that this renaming process has to be performed before task assignments.)

Algorithm 5.1. CONVERT-TABLEAU (t_1, t_2, p):
Input: an existing agent tableau t_1 of shape $\lambda \vdash n$, where $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_j)$.
Output: the standard agent tableau t_2 of shape $\lambda \vdash n$, where $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_j)$; permutation $p \in \mathfrak{S}_n$.

---

[1]The reader is encouraged to compare the definition of a standard agent tableau with that of a *standard Young tableau* [33]. The latter is defined to be a Young tableau whose entries increase in each row and each column [33].

- Write the entries of agent IDs in t_1 in the one-line permutation notation that is obtained by writing entries of t_1 in a sequential order starting from the top left and ending at the bottom right. Call the corresponding permutation for the obtained one-line permutation notation as $p \in \mathfrak{S}_n$.
- Replace t_1 with the standard agent tableau t_2 of the same shape. Note that the corresponding permutation of t_2 for the one-line permutation notation is the identity permutation.
- Return t_2 and permutation $p \in \mathfrak{S}_n$. (Permutation $p \in \mathfrak{S}_n$ might be used for the later purposes if the original agent IDs are not immaterial and need to be restored.)

Now, consider the assignment tableau s_λ in Figure 3(a). Since the agent tableau in s_λ is a standard agent tableau, it follows that s_λ is a standard assignment tableau, i.e., $s_\lambda = S_\lambda$. Thus, we have $S_\lambda = s_\lambda \overset{\text{set}}{=} \{(5 \mapsto 1), (3 \mapsto 2), (1 \mapsto 3), (6 \mapsto 4), (4 \mapsto 5), (2 \mapsto 6)\}$ for $\lambda = (3, 2, 1)$. For a standard assignment tableau, we simply denote S_λ as (t_λ) rather than denoting a 2-tuple (t_λ, a_λ). By a slight abuse of notation, if no confusion arises, we simply denote S_λ as t_λ without parentheses. For instance, Figure 3(c) represents a task assignment $S_\mu \overset{\text{set}}{=} \{(2 \mapsto 1), (3 \mapsto 2), (7 \mapsto 3), (8 \mapsto 4), (6 \mapsto 5), (1 \mapsto 6), (5 \mapsto 7), (4 \mapsto 8)\}$ for $\mu = (4, 2, 2)$, where eight agents are partitioned into three agent groups for $\mu = (4, 2, 2)$.

When we consider an assignment tableau, the partition constraints mandate that both task and agent tableau have the same shape. Once the agent tableau has chosen for an assignment tableau of shape λ, we may fix the agent tableau and consider the permutations of task tableaux of shape λ in order to find other n-task-n-agent assignments. A standard assignment tableau is the preferred form for an n-task-n-agent assignment because a single tableau rather than a 2-tuple of Young tableaux represents an n-task-n-agent assignment between tasks and agents.

We next describe a row-equivalence class of task, agent, and assignment tableaux.

Definition 5.4. An *agent tabloid* of shape λ, denoted $\{a_\lambda\}$, is a row-equivalence class of agent tableaux, i.e., $\{a_\lambda\} = \{a'_\lambda : a'_\lambda \sim a_\lambda\}$, such that agents in the same row of a_λ have the same execution rate. A *task tabloid* of shape λ, denoted $\{t_\lambda\}$, is a row-equivalence class of task tableaux, i.e., $\{t_\lambda\} = \{t'_\lambda : t'_\lambda \sim t_\lambda\}$.

Agents that have the same execution rate are equivalent up to the n-task-n-agent assignment problem in terms of the task turnaround time. For instance, suppose that we have two distinct tasks a and b, and two agents x and y that have the same execution rate. Then, 2-task-2-agent assignments $\{(a \mapsto x), (b \mapsto y)\}$, and $\{(a \mapsto y), (b \mapsto x)\}$ are equivalent in terms of the task turnaround time and the average agent utilization. In Figure 1(b), the execution rates of agents in each set $\{1, 2, 3\}$, $\{4, 5, 6\}$, and $\{7, 8\}$ are the same, respectively. In this case, we may represent them as an agent tabloid that has the entries of the first row 1, 2, and 3, the entries of the second row 4, 5, and 6, and the entries of the third row 7, and 8, respectively.

Definition 5.5. An assignment tabloid of shape λ, denoted $\{s_\lambda\}$, is defined as a 2-tuple of $\{s_\lambda\} \overset{\text{def}}{=} (t_\lambda, \{a_\lambda\}) \overset{\text{def}}{=} \{(t_\lambda, a'_\lambda) : a'_\lambda \sim a_\lambda\}$. Equivalently, $\{s_\lambda\} \overset{\text{def}}{=} (\{t_\lambda\}, a_\lambda) \overset{\text{def}}{=} \{(t'_\lambda, a_\lambda) : t'_\lambda \sim t_\lambda\}$. If an agent tableau is a standard agent tableau or an agent tabloid containing a standard agent tableau, the associated assignment tabloid is said to be *standard*, denoted $\{S_\lambda\} \overset{\text{def}}{=} (\{t_\lambda\})$. By a slight abuse of notation, if no confusion arises, we also denote $\{S_\lambda\}$ as $\{t_\lambda\}$ (without parentheses).

Since both definitions in Definition 5.5 involve a row-equivalence class of Young tableaux of the same shape, the entry order in the same row within an assignment tableau is irrelevant, i.e., they are equivalent up to the n-task-n-agent assignment problem. In

case an agent tabloid is given instead of an agent tableau, we replace the agent tabloid with the corresponding agent tableau, and the task tableau with the corresponding task tabloid in order to keep the canonical form of an assignment tabloid. For instance, if $t_\lambda = \begin{array}{|c|c|}\hline 3 & 1 \\\hline 2 \\\cline{1-1}\end{array}$ and $\{a_\lambda\} = \dfrac{\boxed{1\ \ 2}}{\boxed{3}}$, then $\{s_\lambda\} = \left(\begin{array}{|c|c|}\hline 3 & 1 \\\hline 2 \\\cline{1-1}\end{array}, \dfrac{\boxed{1\ \ 2}}{\boxed{3}} \right)$. Equivalently, the above $\{s_\lambda\}$ can be written as $\{s_\lambda\} = \left(\dfrac{\boxed{1\ \ 3}}{\boxed{2}}, \begin{array}{|c|c|}\hline 1 & 2 \\\hline 3 \\\cline{1-1}\end{array} \right)$. We see that $\{s_\lambda\}$ is a standard assignment tabloid. Thus, $\{s_\lambda\} = \{S_\lambda\} = \dfrac{\boxed{1\ \ 3}}{\boxed{2}}$.

Lemma 5.1. *Suppose $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_i) \vdash n$. For a given assignment tabloid of shape λ, the number of n-task-n-agent assignments represented by the given assignment tabloid is $\lambda_1! \lambda_2! \cdots \lambda_i!$.*

Proof. Let $\{s_\lambda\}$ be an assignment tabloid such that $\{s_\lambda\} = (\{t_\lambda\}, a_\lambda)$. Then, the number of n-task-n-agent assignments represented by $\{s_\lambda\}$ for $\lambda \vdash n$ corresponds to the number of distinct task tableaux in $\{t_\lambda\}$. Therefore, the conclusion follows from Lemma 3.1. $\square$

Lemma 5.2. *Suppose $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_i) \vdash n$. Then, the number of distinct standard assignment tabloids of shape λ is $n!/(\lambda_1! \lambda_2! \cdots \lambda_i!)$.*

Proof. It immediately follows from Lemma 3.2. $\square$

For instance, consider standard assignment tabloids of the following shapes $\lambda_i \vdash n$ for $n = 4$ and $1 \le i \le 3$:

$$\lambda_1 = (4) : \begin{array}{|c|c|c|c|}\hline & & & \\\hline\end{array}, \qquad \lambda_2 = (1,1,1,1) : \begin{array}{|c|}\hline \\\hline \\\hline \\\hline \\\hline\end{array}, \qquad \lambda_3 = (3,1) : \begin{array}{|c|c|c|}\hline & & \\\hline \\\cline{1-1}\end{array}.$$

The number of distinct standard assignment tabloids of the shape λ_1 is $(n!/\lambda_1!) = (4!/4!) = 1$ for $n = 4$. This situation may arise if all four agents are homogeneous. The number of distinct standard assignment tabloids of the shape λ_2 is $(n!/(1!)^n) = n! = 4!$ for $n = 4$. It corresponds to the number of all permutations of four tasks for a given standard agent tableau of shape λ_2. The number of distinct standard assignment tabloids of the shape λ_3 is $n!/(n-1)! = 4$ for $n = 4$, which corresponds to the number of choices (from 1 to 4) for the element in the second row.

Proposition 5.1. *Let $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_i) \vdash n$. If $\{s_\lambda\}$ is an assignment tabloid such that $\{s_\lambda\} = (\{t_\lambda\}, a_\lambda)$, then n-task-n-agent assignments represented by $\{s_\lambda\}$ have the same task turnaround time (respectively, average agent utilization).*

Proof. By Definition 5.5, we have $\{s_\lambda\} = (\{t_\lambda\}, a_\lambda) = (t_\lambda, \{a_\lambda\}) = \{(t_\lambda, a'_\lambda) : a'_\lambda \sim a_\lambda\}$. Suppose to the contrary that the conclusion does not hold. Then, there exists two assignment tableaux $s^1_\lambda = (t_\lambda, a^1_\lambda) \in \{s_\lambda\}$ and $s^2_\lambda = (t_\lambda, a^2_\lambda) \in \{s_\lambda\}$ such that the task turnaround time (respectively, average agent utilization) of task assignments represented by s^1_λ and s^2_λ are not the same. It follows that there exists task t, and two agents a and b in the same row of a^1_λ and a^2_λ, such that the task execution time of t on a and t on b necessarily differs, which is impossible by the choice of a and b since the execution rate of a and b are the same by Definition 5.4. Thus, we conclude that n-task-n-agent assignments represented by $\{s_\lambda\}$ have the same task turnaround time (respectively, average agent utilization). $\square$

Proposition 5.2. *Let* $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_i) \vdash n$. *If* $\{S_\lambda\}$ *is a standard assignment tabloid such that* $\{S_\lambda\} = \{t_\lambda\}$, *then n-task-n-agent assignments represented by* $\{S_\lambda\}$ *have the same task turnaround time (respectively, average agent utilization).*

Proof. It immediately follows from Proposition 5.1. □

By Proposition 5.2, the search space of the n-task-n-agent assignment problem can be reduced to the set of distinct standard assignment tabloids of a given shape $\lambda \vdash n$ instead of $\mathfrak{S}_n$. Note that the converse of Proposition 5.1 and 5.2 is not necessarily true. Two n-task-n-agent assignments represented by two different assignment tabloids may have the same task turnaround time.

(a)

(b)

T: Task, C: Computation requirements (τ)
A: Agent, E: Execution rate (τ/u)

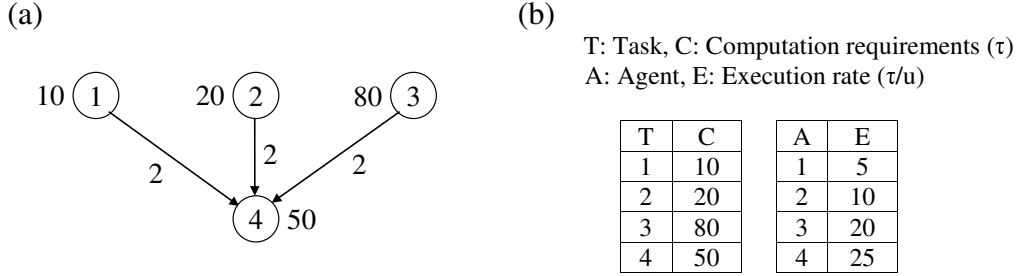| T	C		A	E
1	10		1	5
2	20		2	10
3	80		3	20
4	50		4	25

Figure 4. (a) Task graph; (b) characteristics of tasks, agents.

Suppose we have four tasks and four heterogeneous agents as shown in Figure 4. Then, a standard assignment tabloid of shape $(1, 1, 1, 1)$ represents a n-task-n-agent assignment for $n = 4$ in a unique manner. For instance, consider two different standard assignment tabloids of shape $(1, 1, 1, 1)$ having the task assignment sets $A_1 \overset{\text{set}}{=} \{(1 \mapsto 1), (2 \mapsto 2), (3 \mapsto 3), (4 \mapsto 4)\}$ and $A_2 \overset{\text{set}}{=} \{(1 \mapsto 2), (2 \mapsto 1), (3 \mapsto 3), (4 \mapsto 4)\}$. A simple calculation shows that A_1 and A_2 have the same task turnaround time (8 time units) although their standard assignment tabloids are different.

Standard assignment tabloids are further studied in Section 6, in which we consider the cases when tasks are reassigned for n-task-n-agent assignments represented by standard assignment tabloids of a given shape $\lambda \vdash n$.

5.2 *Generalized assignment tableaux for n-task-m-agent assignments*

An assignment tableau represents a task assignment, where each task in a cell (i, j) of a task tableau is assigned to each agent in a cell (i, j) of an agent tableau bijectively. If a set of tasks is assigned to a smaller-sized set of agents, i.e., n-task-m-agent assignment $(n > m)$, we use generalized Young tableaux to represent n-task-m-agent assignments $(n > m)$.

Definition 5.6. A *generalized agent tableau* of shape λ, denoted $\bar{a}_\lambda$, is a filling of the *Young diagram* of shape λ with agents $\{1, 2, \ldots, n\}$ (repetitions allowed).

Definition 5.7. A *generalized assignment tableau* of shape λ, denoted $\bar{s}_\lambda$, is a 2-tuple of Young tableaux $\bar{s}_\lambda \overset{\text{def}}{=} (t_\lambda, \bar{a}_\lambda)$, where t_λ is a task tableau of shape λ and $\bar{a}_\lambda$ is a generalized agent tableau of shape λ.

Definition 5.8. A *standard task tableau* of shape λ, denoted T_λ, is a task tableau having the entries of task IDs $\{1, 2, \ldots, n\}$ in a sequential order, starting from the top left and ending at the bottom right. If a task tableau is the standard task tableau, then we say that the associated generalized assignment tableau is *standard*, denoted $\bar{S}_\lambda$, i.e., $\bar{S}_\lambda \overset{\text{def}}{=} (T_\lambda, \bar{a}_\lambda)$.

(a) $\bar{s}_\lambda = \left(\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 & 5 \\ \cline{1-2} 6 \\ \cline{1-1} \end{array} \;,\; \begin{array}{|c|c|c|} \hline 1 & 1 & 2 \\ \hline 1 & 3 \\ \cline{1-2} 2 \\ \cline{1-1} \end{array} \right)$ (b) $\bar{S}_\lambda = \left(\begin{array}{|c|c|c|} \hline 1 & 1 & 2 \\ \hline 1 & 3 \\ \cline{1-2} 2 \\ \cline{1-1} \end{array} \right)$ (c) $\bar{S}_\lambda = \begin{array}{|c|c|c|} \hline 1 & 1 & 2 \\ \hline 1 & 3 \\ \cline{1-2} 2 \\ \cline{1-1} \end{array}$

Figure 5. (a) Generalized assignment tableau; (b),(c) standard generalized assignment tableaux.

Consider the generalized assignment tableau $\bar{s}_\lambda$ in Figure 5(a). Since the task tableau in $\bar{s}_\lambda$ is a standard task tableau, $\bar{s}_\lambda$ is a standard generalized assignment tableau, i.e., $\bar{s}_\lambda = \bar{S}_\lambda$ by Definition 5.8. Thus, we have $\bar{s}_\lambda = \bar{S}_\lambda \stackrel{\text{set}}{=} \{(1 \mapsto 1), (2 \mapsto 1), (3 \mapsto 2), (4 \mapsto 1), (5 \mapsto 3), (6 \mapsto 2)\}$ for $\lambda = (3, 2, 1)$. As shown in Figure 5(b), we also denote $\bar{S}_\lambda$ as $(\bar{a}_\lambda)$ rather than denoting a 2-tuple $(t_\lambda, \bar{a}_\lambda)$ for a standard generalized assignment tableau. Similarly to a standard assignment tableau, by a slight abuse of notation, we may denote $\bar{S}_\lambda$ as $\bar{a}_\lambda$ without parentheses if no confusion arises (see Figure 5(c)).

Note that a shape λ of a standard assignment tableau $S_\lambda := (t_\lambda, A_\lambda)$ (i.e., $S_\lambda := t_\lambda$) can be given based on a logical partition of agents. Meanwhile, a shape λ of a standard generalized assignment tableau $\bar{S}_\lambda := (T_\lambda, \bar{a}_\lambda)$ (i.e., $\bar{S}_\lambda := \bar{a}_\lambda$) can be given based on a logical partition of tasks rather than agents. Unlike n-task-n-agent assignments, a standard generalized assignment tableau alone does not necessarily determine the task turnaround time of a task assignment. As mentioned in Section 2.3, the start time of each task for an n-task-n-agent assignment is its unique data ready time. Meanwhile, the start time of each task for an n-task-m-agent assignment for $n > m$ depends on an execution order, i.e., temporal assignment of tasks. For instance, if agent a has two tasks t_1 and t_2 with the same data ready time, then it depends on a task assignment algorithm to determine whether t_1 or t_2 starts first on a. However, if no task precedence constraint is given, we may obtain the task turnaround time of a given standard generalized assignment tableau using the $|T| \times |A|$ cost matrix, where $|T|$ is the number of tasks and $|A|$ is the number of agents.

In Section 5.1 we showed that the search space of the n-task-n-agent assignment problem can be reduced to the set of standard assignment tabloids of a given shape $\lambda \vdash n$. We now consider a search space for the n-task-m-agent assignment problem $(n > m)$ consisting of standard generalized assignment tableaux of a given shape. Let $X_j^\lambda = \{t_1, t_2, \ldots, t_j\}$ denote a set of distinct standard generalized assignment tableaux $t_i (1 \le i \le j)$ of a given shape λ. If $j = n^m$, then X_j^λ is the whole search space of the n-task-m-agent assignment problem represented by standard generalized assignment tableaux of a given shape λ (see the following "Remarks"). If $j = k$ for $k < n^m$, it is a selected search space of the n-task-m-agent assignment problem represented by standard generalized assignment tableaux of a given shape λ.

Remarks. The number of distinct standard general assignment tableaux of shape $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_i) \vdash n$ for m agents is n^m. Using an elementary counting argument, we see that the number of distinct standard generalized assignment tableaux of shape $\lambda \vdash n$ for m agents is the same with the number of m permutations from the set of n tasks with repetition allowed, which is n^m. Note that it does not rely on a given shape of standard generalized assignment tableaux.

The following proposition describes an equivalence relation on a set of standard generalized assignment tableaux of a given shape with respect to task reassignments defined by a group action.

Proposition 5.3. *Let $G \le \mathfrak{S}_n$ act on $X_n^\lambda = \{t_1, t_2, \ldots, t_n\}$ by $gt_i = t_{g(i)}$ for $g \in G$. For $t_1, t_2 \in X_n^\lambda$, let $t_1 \sim_{tr} t_2$ iff there exists $g \in G$ such that $gt_1 = t_2$. Then, $\sim_{tr}$ is an equivalence relation on X_n^λ.*

Proof. Since $et = t$ for each $t \in X_n^\lambda$, we have $t \sim_{tr} t$. Thus, $\sim_{tr}$ is reflexive.

To show that $\sim_{tr}$ is symmetric, assume $t_1 \sim_{tr} t_2$ for $t_1, t_2 \in X_n^\lambda$. It follows that $gt_1 = t_2$ for some $g \in G$. Now, we have $g^{-1}(gt_1) = g^{-1}(t_2)$. Since $g^{-1}(t_2) = g^{-1}(gt_1) = et_1 = t_1$, we have $t_2 \sim_{tr} t_1$. Thus, $\sim_{tr}$ is symmetric.

To show that $\sim_{tr}$ is transitive, assume that $t_1 \sim_{tr} t_2$ and $t_2 \sim_{tr} t_3$ for $t_1, t_2, t_3 \in X_n^\lambda$. Then, we have $g_1 t_1 = t_2$ and $g_2 t_2 = t_3$ for some $g_1, g_2 \in G$. We claim that $(g_2 g_1) t_1 = t_3$, which shows that $t_1 \sim_{tr} t_3$. Since $(g_2 g_1) t_1 = g_2(g_1 t_1) = g_2(t_2) = t_3$, we have $(g_2 g_1) t_1 = t_3$. Thus, $t_1 \sim_{tr} t_3$, which shows that $\sim_{tr}$ is transitive. $\square$

The following proposition describes the size of the equivalence class of $t \in X_n^\lambda$ with respect to the equivalence relation $\sim_{tr}$. We first denote the equivalence class of $t \in X_n^\lambda$ with respect to $\sim_{tr}$ by $G(t) = \{gt : g \in G\}$ when $G \leq \mathfrak{S}_n$ act on $X_n^\lambda = \{t_1, t_2, \ldots, t_n\}$ by $gt_i = t_{g(i)}$ for $g \in G$.

Proposition 5.4. *Let $G \leq \mathfrak{S}_n$ act on $X_n^\lambda = \{t_1, t_2, \ldots, t_n\}$ as above and let $G^t = \{g \in G : gt = t\}$ for $t \in X_n^\lambda$. The size of the equivalence class of $t \in X_n^\lambda$ with respect to the equivalence relation $\sim_{tr}$ in Proposition 5.3 is $|G|/|G^t|$.*

Proof. We show that $gG^t \mapsto gt$ is a well-defined bijection from the set of cosets of G^t in G onto the equivalence class $G(t) = \{gt : g \in G\}$ for $t \in X_n^\lambda$. We first show that G^t is a subgroup of G.

Let $a, b \in G^t$. Then, we have $at = t$ and $bt = t$. It follows that $(ab)t = a(bt) = at = t$. Thus, $ab \in G^t$, which shows that G^t is closed under the binary operation of G. Since $et = t$, we have $e \in G^t$ as well. Let $h \in G^t$. Then, we have $ht = t$. Since $t = et = (h^{-1}h)t = h^{-1}h(t) = h^{-1}t$, it follows that $h^{-1} \in G^t$. Thus, G^t is a subgroup of G.

Let $g_1, g_2 \in G$. Since $g_1 t = g_2 t \iff g_2^{-1} g_1 t = t \iff g_2^{-1} g_1 \in G^t \iff g_1 G^t = g_2 G^t$, we see that $gG^t \mapsto gt$ is a well-defined bijection.

Thus, the size of the equivalence class of $t \in X_n^\lambda$ with respect to the equivalence relation $\sim_{tr}$ in Proposition 5.3 is $[G : G^t] = |G|/|G^t|$. $\square$

Proposition 5.4 directly follows from the *orbit-stabilizer theorem* [19] in group theory. The interested reader may refer to [13, 19, 21] for further details.

Now, we illustrate how Proposition 5.4 applies to a simple search space consisting of standard generalized assignment tableaux of a given shape. We assume that task reassignments are closed with respect to a search space, i.e., if a task reassignment transforms a standard generalized assignment tableau t_i to t_j, then both t_i and t_j belong to the given search space consisting of standard generalized assignment tableaux of a given shape.

For instance, let $G = \{e, (2\,3\,5\,6), (2\,5)(3\,6), (2\,6\,5\,3)\}$ be a subgroup of $\mathfrak{S}_8$ and let $X_8^\lambda = \{t_1, t_2, \ldots, t_8\}$ be a set of distinct standard generalized assignment tableaux of a given shape λ. Let G act on X_8 as above. Since $G^{t_2} = G^{t_3} = G^{t_5} = G^{t_6} = \{e\}$, the size of the equivalence class of each t_2, t_3, t_5, and t_6 is 4 by Proposition 5.4. Similarly, $G^{t_1} = G^{t_4} = G^{t_7} = G^{t_8} = G$. It follows that the size of the equivalence class of each t_1, t_4, t_7, and t_8 is $[G : G] = 1$.

In contrast, the reader is encouraged to verify that if $\mathfrak{S}_8$ acts on X_8^λ as above, the equivalence class of each t_i for $1 \leq i \leq 8$ is the whole X_8^λ.

Now, we illustrate an algorithm for finding the equivalence class of a standard generalized assignment tableau of a given shape with respect to $\sim_{tr}$. The following algorithms are slight modifications of the *orbit-stabilizer algorithms* discussed in [19, 20][1].

We first convert X_n^λ into a totally ordered set $(X_n^\lambda, \leq_t)$ by assigning each $t_i \in X_n^\lambda$ to

---

[1] In computational group theory right group actions are often considered [19, 20] for convenience. However, we are only concerned with left group actions and stick with them throughout this paper.

a number, denoted $|t_i|$, which is the sum of all entries of t_i [30]. The total order $\leq_t$ is defined on X_n^λ in such a way that $t_i \leq_t t_j$ iff $|t_i| \leq |t_j|$. Then, we rename each tableau of X_n^λ in such a manner that $t_1 \leq_t t_2 \leq_t \cdots \leq_t t_n$.

Algorithm 5.2. TASK-REASSIGNMENT-ORBIT $(t_j, X_n^\lambda, \Omega)$:
Input: a standard generalized assignment tableau $t_j \in X_n^\lambda$ of shape $\lambda \vdash k$, a permutation group $G \leq \mathfrak{S}_n$ given by a generating set $\Omega = \{g_1, \ldots, g_m\}$.
Output: the equivalence class of a given standard generalized assignment tableau t_j with respect to $\sim_{tr}$.

```
1    Δ := [t_j];
2    for α ∈ Δ,  g ∈ Ω do
3        β = gα;
3        if  β ∉ Δ    then
4            append β to Δ;
5    return Δ;
```

In Algorithm 5.2 a permutation group is given by the set of m generators. Therefore, there will be $|\Delta|m$ images to compute.

As an example of Algorithm 5.2, consider $(X_{50}^\lambda, <_t)$, where $X_{50}^\lambda = \{t_1, t_2, \ldots, t_{50}\}$ and $t_1 <_t t_2 <_t \cdots <_t t_{50}$ for standard generalized assignment tableaux $t_i (1 \leq i \leq 50)$ of shape $\lambda \vdash k$. (We write $t_i <_t t_j$ if $t_i \leq_t t_j$ and $|t_i| \neq |t_j|$.) If $G \leq \mathfrak{S}_{50}$ act on $X_{50}^\lambda = \{t_1, t_2, \ldots, t_{50}\}$ by $gt_i = t_{g(i)}$ for $g \in G$ as discussed and a group G is generated by $\Omega = \{(1\,2), (2\,3), (3\,4)\}$, then the equivalence class of t_1 with respect to $\sim_{tr}$ is $\{t_1, t_2, t_3, t_4\}$. On the other hand, if a group is generated by $\Omega' = \{(47\,48), (48\,49), (49\,50)\}$, then the equivalence class of t_{48} with respect to $\sim_{tr}$ is $\{t_{47}, t_{48}, t_{49}, t_{50}\}$. A typical example of a standard generalized assignment tableau in $\{t_1, t_2, t_3, t_4\}$ is one with smaller number entries from $\{1, 2, \ldots, k\}$, while a typical example of a standard generalized assignment tableau in $\{t_{47}, t_{48}, t_{49}, t_{50}\}$ is one with larger number entries from $\{1, 2, \ldots, k\}$.

Let $G(t_j)$ denote the equivalence class of $t_j \in X_n^\lambda$ with respect to $\sim_{tr}$. For each $\sigma \in G(t_j)$, the following algorithm finds an element $g \in G$ such that $gt_j = \sigma$. It returns two lists Δ and L, in which Δ stores each $\sigma \in G(t_j)$, while L stores the corresponding group element $g \in G$ satisfying $gt_j = \sigma$. The first elements of Δ and L are t_j and e, respectively. In Algorithm 5.3 $L[\sigma]$ denotes $L[i]$ where $\Delta[i] = \sigma$. Note that if we have two arbitrary $\sigma_1, \sigma_2 \in G(t_j)$, we can find $g \in G$ such that $g\sigma_1 = \sigma_2$ using the following algorithm by obtaining $g_1 t_j = \sigma_1$ and $g_2 t_j = \sigma_2$ (i.e., $g = g_2 g_1^{-1}$).

Algorithm 5.3. TASK-REASSIGNMENT-TRANSVERSAL $(t_j, X_n^\lambda, \Omega)$:
Input: a standard generalized assignment tableau $t_j \in X_n^\lambda$ of shape $\lambda \vdash k$, a permutation group $G \leq \mathfrak{S}_n$ given by a generating set $\Omega = \{g_1, \ldots, g_m\}$.
Output: Transversal L, the equivalence class of a given standard generalized assignment tableau t_j with respect to $\sim_{tr}$.

```
1    Δ := [t_j];
2    L := [e];
3    for α ∈ Δ do
4        for i ∈ {1, ..., m} do
5            β = g_i α;
6                if  β ∉ Δ then
```

7		append β to Δ;
8		append $g_i \cdot L[\alpha]$ to L;
9	**return** L, Δ;	

For instance, consider when $G \leq \mathfrak{S}_{50}$ acts on $X_{50}^\lambda = \{t_1, t_2, \ldots, t_{50}\}$ by $gt_i = t_{g(i)}$ for $g \in G$ and a group G is generated by $\Omega = \{g_1, g_2, g_3\}$, where $g_1 = (1\,2)$, $g_2 = (2\,3)$, and $g_3 = (3\,4)$. As seen above, the equivalence class of t_1 (i.e., $G(t_1)$) with respect to $\sim_{tr}$ is $\{t_1, t_2, t_3, t_4\}$. Now, we find a group element $g \in G$ such that $gt_1 = t_4$ using Algorithm 5.3. The first elements of Δ and L are given as t_1 and e, respectively (i.e., $\Delta[1] = t_1$ and $L[1] = e$). The reader is encouraged to verify that $\Delta[2] = t_2$, $L[2] = (1\,2)$, $\Delta[3] = t_3$, $L[3] = (2\,3)(1\,2)$, $\Delta[4] = t_4$, and $L[4] = (3\,4)(2\,3)(1\,2)$. Since $\Delta[4] = t_4$, we have $g = L[4] = (3\,4)(2\,3)(1\,2)$.

The time-complexity of both Algorithm 5.2 and Algorithm 5.3 is proportional to $|\Delta|m$. To test whether $\beta \in \Delta$ or $\beta \notin \Delta$ for both algorithms is basically a search problem. An additional data structure (e.g., list) can be used for Δ to maintain the sorted hash value for each entry t_i in Δ.

The correctness of Algorithms 5.2 and 5.3 directly follows from the *orbit algorithms* [19, 20] used in GAP [17] by renaming the elements of X_n^λ such that $t_i \mapsto i$ for $t_i \in X_n^\lambda$ and defining the action of $G \leq \mathfrak{S}_n$ on the resulting set, denoted $\overline{X}_n^\lambda = \{1, 2, \ldots, n\}$, simply by $gi = g(i)$ for $g \in G, i \in \overline{X}_n^\lambda$ accordingly. The total order $\leq$ is naturally defined on the set $\overline{X}_n^\lambda$. (The orbit algorithms have already been well-established when $G \leq \mathfrak{S}_n$ acts on the set $\{1, 2, \ldots, n\}$ as above [17]. In Appendix B we discuss how GAP is used for the applications of Propositions 5.3 and 5.4.)

Note that the total order defined on X_n^λ in this section is not the only total order that can be defined on X_n^λ. The purpose of defining an equivalence relation and a total order on X_n^λ is to explore the search space corresponding to X_n^λ in a well-defined and systematic manner by means of algebraic operations (e.g., a group action). We leave it as an open question to consider and define other useful total orders on X_n^λ that exploits the symmetry during the navigation of a search space.

6. Task reassignments on the set of standard assignment tabloids of shape $\lambda \vdash n$

The counting aspects[1] of the search space of the n-task-n-agent assignment problem using tabloids of a given shape are further investigated in this section.

We first give a brief overview of the necessary background of this section. We assume that the reader has some familiarity with a *vector space* [13] over a *field* $\mathbb{K}$ [9]. Examples of fields are the set of real numbers $\mathbb{R}$ and the set of complex numbers $\mathbb{C}$. In the remainder of this section, G denotes a finite group, $\mathbb{K}$ a field, and V denotes a finite dimensional vector space. Definitions and results used in the following overview are found in [1, 9, 13, 16, 21, 24, 31, 33].

Suppose G acts on V over $\mathbb{K}$. The action of G on V is called *linear* if the following conditions are met: (i) $g(v + w) = gv + gw$ for all $g \in G$ and $v, w \in V$, (ii) $g(kv) = k(gv)$ for all $g \in G, k \in \mathbb{K}$, and $v \in V$. If G acts on V linearly, then V is called a *G-module*.

Let $(G, \cdot)$ and $(G', \circ)$ be groups. A map $\phi: G \to G'$ is a *homomorphism* if $\phi(x \cdot y) = \phi(x) \circ \phi(y)$ for all $x, y \in G$.

---

[1]The counting argument of task assignments in distributed systems was researched in [37]. However, its scope is quite different from that of our approach.

The *general linear group* $\mathrm{GL}(n, \mathbb{K})$ is the group of all invertible $n \times n$ matrices over $\mathbb{K}$, whose binary operation is matrix multiplication.

A *matrix representation* of G is any homomorphism from G into $\mathrm{GL}(n, \mathbb{K})$.

The *trace* of an $n \times n$ matrix $A = (a_{i,j})$ over $\mathbb{K}$, denoted $\mathrm{tr}(A)$, is defined to be $a_{1,1} + a_{2,2} + \cdots + a_{n,n} \in \mathbb{K}$.

If X is a matrix representation of G, then the *character* of X is a function $\chi : G \to \mathbb{K}$ defined by $\chi(g) = \mathrm{tr} X(g)$ for any $g \in G$. If V is a G-module, then its character, denoted χ_V, is the character of a matrix representation X of G corresponding to V.

Let $S = \{x_1, x_2, \ldots, x_n\}$ and let V be a vector space over $\mathbb{K}$ with basis $\{e_x : x \in S\}$. Let G act on V by $g(\sum k_x e_x) = \sum k_x e_{gx}$ for $g \in G$ and $k_x \in \mathbb{K}$ by linearly extending the action of G on S. The G-module V is called the *permutation module* of G on the set S.

For instance, consider the permutation module of $\mathfrak{S}_3$ associated with the set $S = \{1, 2, 3\}$. Let V be an 3-dimensional vector space over $\mathbb{K}$ with basis $B = \{e_1, e_2, e_3\}$. We give V an $\mathfrak{S}_3$-module structure by defining $g(\sum k_x e_x) = \sum k_x e_{gx}$ for any $g \in \mathfrak{S}_3$ and $k_x \in \mathbb{K}$ in an obvious way. For instance, $(1\,2)e_1 = e_2$, $(1\,2)e_2 = e_1$, and $(1\,2)e_3 = e_3$. The matrix representation $X(g)$ at $g \in \mathfrak{S}_3$ corresponding to V has a 1 in row i and column j if $g \cdot e_j = e_i$, and 0 otherwise.

$$X(e) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, X((1\,2)) = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, X((1\,3)) = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix},$$
$$X((2\,3)) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, X((1\,2\,3)) = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, X((1\,3\,2)) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}.$$

The reader is encouraged to verify that X is indeed a matrix representation of $\mathfrak{S}_3$, e.g., $X((1\,3\,2)) = X((1\,2)(1\,3)) = X((1\,2))X((1\,3))$ and $X(e) = X((1\,2)(1\,2)) = X((1\,2))X((1\,2))$. We see that the value of the character χ_V of V at g equals the number of elements of $B = \{e_1, e_2, e_3\}$ that are fixed by the action of $g \in \mathfrak{S}_3$ in the above example.

Let H be a subgroup of a group G. An action of $h \in H$ on the set G defined by $(h, x) \mapsto hxh^{-1}$ is called *conjugation* by h. If a group G acts on itself by conjugation, then the set $\{gxg^{-1} : g \in G\}$ of $x \in G$ is called the *conjugacy class* of x. Two elements of $\mathfrak{S}_n$ are in the same conjugacy class iff they have the same cycle type.

Theorem 6.1 ([9, 33]). *If K is a conjugacy class of G, then $g, h \in K$ implies $\chi(g) = \chi(h)$.*

Theorem 6.1 says that characters are constant on conjugacy classes.

Theorem 6.2 ([16]). *Let $S = \{x_1, x_2, \ldots, x_n\}$ and let V be the associated permutation module of G on S. The value of the character χ_V of V at $g \in G$ equals the number of elements of S that are fixed by the action of $g \in G$.*

The action of $\pi \in \mathfrak{S}_n$ on a Young tableau $t = (t_{i,j})$ of shape $\lambda \vdash n$ is defined here by $\pi t = (\pi(t_{i,j}))$, where $t_{i,j}$ denotes the entry of t in position (i, j). In a similar manner the action of $\pi \in \mathfrak{S}_n$ on tabloids is defined by $\pi\{t\} = \{\pi t\}$. For instance, $(2\,3) \in \mathfrak{S}_3$ acts on a tabloid of shape $\lambda = (2, 1)$ as shown below:

$$(2\,3) \; \frac{\begin{array}{cc} 1 & 2 \end{array}}{\begin{array}{c} 3 \end{array}} = \frac{\begin{array}{cc} 1 & 3 \end{array}}{\begin{array}{c} 2 \end{array}}$$

We see that $(2\,3) \in \mathfrak{S}_3$ gives a permutation to a tabloid of shape λ, swapping "2" and "3" in the tabloid.

Suppose $\lambda \vdash n$. Let V^λ denote the vector space over the field of real numbers $\mathbb{R}$ whose basis consists of the set of tabloids of shape λ, i.e., $V^\lambda = \mathbb{R}\{\{t_1\}, \ldots, \{t_k\}\}$, where $\{t_1\}, \ldots, \{t_k\}$ is a complete list of distinct tabloids of shape λ. Then, V^λ is a permutation

module of $\mathfrak{S}_n$ on the set of distinct tabloids of shape λ [33].

The following theorem provides a formula to compute the characters of V^λ for $\lambda \vdash n$.

Theorem 6.3 ([50])**.** *Let* $\lambda = (\lambda_1, \ldots, \lambda_i)$ *and* $\mu = (\mu_1, \ldots, \mu_j)$ *be partitions of* n*. The characters of* V^λ *evaluated at an element of* $\mathfrak{S}_n$ *with cycle type* μ *is equal to the coefficient of* $x_1{}^{\lambda_1} x_2{}^{\lambda_2} \cdots x_i{}^{\lambda_i}$ *in*

$$\prod_{k=1}^{j} (x_1^{\mu_k} + x_2^{\mu_k} + \cdots + x_i^{\mu_k}).$$

We now consider task reassignments by means of a group action for n-task-n-agent assignments represented by the set of standard assignment tabloids of shape $\lambda \vdash n$. We define the action of $\pi \in \mathfrak{S}_n$ on standard assignment tabloids in exactly the same way as above.

$$\{S_\mu^1\} = \overline{\begin{array}{ccc} 1 & 2 & 3 \\ 4 \end{array}}, \{S_\mu^2\} = \overline{\begin{array}{ccc} 1 & 2 & 4 \\ 3 \end{array}}, \{S_\mu^3\} = \overline{\begin{array}{ccc} 1 & 3 & 4 \\ 2 \end{array}}, \{S_\mu^4\} = \overline{\begin{array}{ccc} 2 & 3 & 4 \\ 1 \end{array}}.$$

Figure 6. Standard assignment tabloids of shape $\mu = (3, 1)$.

Consider $g = (1\,2) \in \mathfrak{S}_4$ acts on each standard assignment tabloid $\{S_\mu^k\}$ for $1 \le k \le 4$. For instance, $(1\,2)\{S_\mu^3\}$ swaps task 1 and task 2 in the n-task-n-agent assignments represented by $\{S_\mu^3\}$. We see that $g\{S_\mu^1\} = \{S_\mu^1\}$, $g\{S_\mu^2\} = \{S_\mu^2\}$, $g\{S_\mu^3\} = \{S_\mu^4\}$, and $g\{S_\mu^4\} = \{S_\mu^3\}$. Note that swapping elements in the same row of the given tabloid remains the tabloid invariant. We see that g fixes $\{S_\mu^1\}$ and $\{S_\mu^2\}$ only. The following proposition is the main result of this section.

Proposition 6.1. *The number of standard assignment tabloids of shape* $\lambda \vdash n$ *fixed by the (task reassignment) action of* $g \in \mathfrak{S}_n$ *is the character of* V^λ *evaluated on the conjugacy class of* $g \in \mathfrak{S}_n$*.*

Proof. Let $\mathfrak{S}_n$ act on the set S consisting of a complete list of distinct standard assignment tabloids of shape $\lambda \vdash n$ as above. Then, we see that its associated permutation module of $\mathfrak{S}_n$ on S is simply V^λ. The number of standard assignment tabloids of shape $\lambda \vdash n$ fixed by the (task reassignment) action of $g \in \mathfrak{S}_n$ is the character of V^λ at $g \in \mathfrak{S}_n$ by Theorem 6.2. Since characters are constant on conjugacy classes by Theorem 6.1, the number of standard assignment tabloids of shape $\lambda \vdash n$ fixed by the (task reassignment) action of $g \in \mathfrak{S}_n$ is the character of V^λ evaluated on the conjugacy class of $g \in \mathfrak{S}_n$. $\square$

By Proposition 6.1, we see how a task reassignment represented by $g \in \mathfrak{S}_n$ affects the elements of the search space of the n-task-n-agent assignment problem represented by the set of standard assignment tabloids of shape $\lambda \vdash n$. Using Lemma 5.2, the size of the search space represented by the set of standard assignment tabloids of a given shape $\lambda \vdash n$ is obtained. By subtracting the number in Proposition 6.1 from the above size of the search space, we obtain the total number of standard assignment tabloids of a given shape $\lambda \vdash n$ that is not invariant by a task reassignment represented by $g \in \mathfrak{S}_n$. The higher the total number, the more elements of the search space become affected and the more computations for task reassignments are required as a result. Note that by Proposition 5.2, a standard assignment tabloid $\{S_\lambda\}$ fixed by the action of $g \in \mathfrak{S}_n$ has the same task turnaround time before and after the action of $g \in \mathfrak{S}_n$, which does not need a task reassignment by $g \in \mathfrak{S}_n$ at all.

Now, consider Table 1, which shows the list of characters of V^λ for each conjugacy class of $\mathfrak{S}_4$ [50]. The characters of V^λ in Table 1 can also be computed by means of Theorem 6.3. For instance, we obtain $\phi^{(2,2)}$ at $K_{(2,1,1)}$ in Table 1 by computing the

Table 1. Characters of V^λ for $\mathfrak{S}_4$ (ϕ^λ: the character of V^λ, K_μ: the conjugacy class of $\mathfrak{S}_4$ with cycle type μ) [50].

	$K_{(1,1,1,1)}$	$K_{(2,1,1)}$	$K_{(2,2)}$	$K_{(3,1)}$	$K_{(4)}$
$\phi^{(1,1,1,1)}$	24	0	0	0	0
$\phi^{(2,1,1)}$	12	2	0	0	0
$\phi^{(2,2)}$	6	2	2	0	0
$\phi^{(3,1)}$	4	2	0	1	0
$\phi^{(4)}$	1	1	1	1	1

coefficient of $x_1^2 x_2^2$ in $(x_1^2 + x_2^2)(x_1 + x_2)^2$, which is 2. (The interested reader may refer to [33, 50] for further details.) Note that the column of $K_{(1,1,1,1)}$ in Table 1 corresponds to the dimension of V^λ, which indicates the number of distinct standard assignment tabloids of shape $\lambda \vdash n$ for $n = 4$. Since $\phi^{(3,1)}$ at $K_{(2,2)}$ is 0 in Table 1, we see that in Figure 6 the number of standard assignment tabloids of shape $\lambda = (3,1)$ fixed by the (task reassignment) action of $(1\,2)(3\,4) \in \mathfrak{S}_4$ is 0 by Proposition 6.1, where $(1\,2)(3\,4) \in \mathfrak{S}_4$ has cycle type $(2,2)$.

7.   Conclusions

This paper presented a framework for representing task assignments and reassignments in distributed systems using Young tableaux and finite groups focused on finite symmetric groups. We showed that a task assignment with a partition is naturally represented by a Young tableau and that task reassignments are described by means of a group action on a set of Young tableaux.

We introduced a standard assignment tableau (respectively, a standard generalized assignment tableau) to represent an n-task-n-agent assignment (respectively, an n-task-m-agent assignment ($n > m$)) in a visual and compact manner, while representing a logical partition of agents (respectively, tasks) in a distributed system.

We discussed row-equivalence classes of Young tableaux to represent certain equivalence classes of standard assignment tableaux. Then, we discussed the search space reduction using equivalence classes of standard assignment tableaux and showed how task reassignments by means of a group action affect the elements of the search space for the n-task-n-agent assignment problem. We also discussed an equivalence relation and a total order on a set of standard generalized assignment tableaux of a given shape in order to explore the selected search space for the n-task-m-agent assignment problem ($n > m$) in a well-defined and systematic manner by means of group-theoretical methods.

By raising the expressiveness of task assignments, our approach is able to employ some of the known results of Young tableaux and group theory for task assignments and reassignments in distributed systems.

There are a wide variety of tableaux (e.g. *skew tableaux* [33], *oscillating tableaux* [40], etc.) and their algorithms (e.g. *row insertion* [33], *deletion, forward and backward slides* [40], etc.) used in combinatorics and group theory, which have not been discussed in this paper. Variants of assignment tableaux can be considered by means of those tableaux. When applying our approach to certain types of distributed systems, one may need to link additional constraints (e.g. task or node priorities, task dependencies [11], etc.) or data structure to entries or cells in an assignment tableau. One may also need to consider the variants of assignment tableaux to represent the specific kinds of tasks or nodes in certain types of distributed systems. We leave it to future work to consider the possible variants of assignment tableaux along with their algorithms in distributed

systems and to examine them from both theoretical and practical perspectives.

References

[1] J.L. Alperin and R.B. Bell, *Groups and Representations*, Springer, New York, NY (1995).

[2] A. Benoit and Y. Robert, *Mapping pipeline skeletons onto heterogeneous platforms*, Journal of Parallel and Distributed Computing 68 (2008), pp. 790–808.

[3] R. Bettati and J.W.S. Liu, *End-to-End Scheduling to Meet Deadlines in Distributed Systems*, in *Proceedings of the 12th International Conference on Distributed Computing Systems*, Yokohama, Japan, 1992, pp. 452–459.

[4] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems*, SIAM, Philadelphia, PA (2009).

[5] L. Bus and P. Tvrdik, *Distributed Memory Auction Algorithms for the Linear Assignment Problem*, in *Proceedings of 14th IASTED International Conference of Parallel and Distributed Computing and Systems*, Cambridge, MA, 2002, pp. 137–142.

[6] Y.B. Cho, T. Kurokawa, Y. Takefuji, and H.S. Kim, *An O(1) approximate parallel algorithm for the n-task-n-person assignment problem*, in *Proceedings of 1993 International Joint Conference on Neural Networks*, Nagoya, Japan, 1993, pp. 1503–1506.

[7] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to algorithms*, 2nd ed., The MIT Press, Cambridge, MA (2001).

[8] B. Cosenza, G. Cordasco, R.D. Chiara, and V. Scarano, *Distributed load balancing for parallel agent-based simulations*, in *19th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2011*, Ayia Napa, Cyprus, 2011, pp. 62–69.

[9] D. Dummit and R. Foote, *Abstract Algebra*, 3rd ed., John Wiley and Sons Inc., Hoboken, NJ (2004).

[10] K. Efe, *Heuristic models of task assignment scheduling in distributed systems*, Computer 15 (1982), pp. 50–56.

[11] H. El-Rewini and T.G. Lewis, *Distributed and Parellel Computing*, Manning Publications Co., Greenwich, CT (1998).

[12] H. El-Rewini, T.G. Lewis, and H.H. Ali, *Task scheduling in parallel and distributed systems*, Prentice-Hall, Inc., Upper Saddle River, NJ (1994).

[13] J.B. Fraleigh, *A First Course in Abstract Algebra*, Addison-Wesley, Reading, MA (1998).

[14] Free Software Foundation, *GNU C++ Library*, http://gcc.gnu.org/onlinedocs/libstdc++.

[15] W. Fulton, *Young Tableaux: With application to Representation Theory and Geometry*, Cambridge University Press, Cambridge (1997).

[16] W. Fulton and J. Harris, *Representation Theory: A First Course*, Springer, New York, NY (1991).

[17] The GAP Group, *GAP - Groups, Algorithms, Programming - a System for Computational Discrete Algebra, Version 4.7.5* (2014), http://www.gap-system.org/.

[18] S. Hartmann and D. Briskorn, *A survey of variants and extensions of the resource-constrained project scheduling problem*, European Journal of Operational Research 207 (2010), pp. 1–14.

[19] D.F. Holt, B. Eick, and E. O'Brien, *Handbook of computational group theory*, CRC Press, Boca Raton, FL (2005).

[20] A. Hulpke, *Notes on computational group theory*, Tech. rep., Colorado State University, Fort Collins, CO, 2010.

[21] T. Hungerford, *Algebra*, Springer, New York, NY (1980).

[22] E. Ilavarasan, P. Thambidurai, and R. Mahilmannan, *High Performance Task Scheduling Algorithm for Heterogeneous Computing System*, in *ICA3PP 2005*, M. Hobbs, A. Goscinski, and W. Zhou, eds., LNCS 3719, Springer, Melbourne, Australia, 2005, pp. 193–203.

[23] M.W. Jang and G. Agha, *Adaptive agent allocation for massively multi-agent applications*, in *Massively Multi-Agent Systems I*, T. Ishida, L. Gasser, and H. Nakashima, eds., LNCS 3446, Springer, Kyoto, Japan, 2005, pp. 25–39.

[24] D. Kim, *Task swapping networks in distributed systems*, International Journal of Computer Mathematics 90 (2013), pp. 2221–2243.

[25] A. Knutson, E. Miller, and A. Yong, *Tableau complexes*, Israel Journal of Mathematics 163 (2008), pp. 317–343.

[26] S. Kumar and P. Jadon, *A Novel Hybrid Algorithm for Permutation Flow Shop Scheduling*, International Journal of Computer Science and Information Technologies 5 (2014), pp. 5057–5061.

[27] J.C. Lin, *Optimal Task Assignment with Precedence in Distributed Computing Systems*, Information Sciences 78 (1994), pp. 1–18.

[28] L. Liu and D.A. Shell, *A Distributable and Computation-flexible Assignment Algorithm: From Local Task Swapping to Global Optimality*, in *2012 Robotics: Science and Systems Conference (RSS)*,

Sydney, Australia, 2012.

[29] A. Naiem and M. El-Beltagy, *Deep Greedy Switching: A Fast and Simple Approach For Linear Assignment Problems*, in *7th International Conference of Numerical Analysis and Applied Mathematics (ICNAAM 2009)*, Rethymno, Greece, 2009.

[30] V. Prosper, *Factorization properties of the q-specialization of Schubert polynomials*, Annals of Combinatorics 4 (2000), pp. 91–107.

[31] J.J. Rotman, *The Theory of Groups: An Introduction*, Allyn and Bacon, Inc, Boston, MA (1965).

[32] J.E. Rowe, M.D. Vose, and A.H. Wright, *Group properties of crossover and mutation*, Evolutionary Computation 10 (2002), pp. 151–184.

[33] B.E. Sagan, *The Symmetric Group: Representations, Combinatorial Algorithms, and Symmetric Functions*, 2nd ed., Springer, New York, NY (2001).

[34] A. Sahu and R. Tapadar, *Solving the Assignment problem using Genetic Algorithm and Simulated Annealing*, IAENG International Journal of Applied Mathematics 36 (2007), pp. 37–40.

[35] A.Z.S. Shahul and O. Sinnen, *Optimal Scheduling of Task Graphs on Parallel Systems*, in *Proceedings of the 2008 Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies*, Dunedin, New Zealand, 2008, pp. 323–328.

[36] C.C. Shen and W.H. Tsai, *A Graph Matching Approach to Optimal Task Assignment in Distributed Computing System Using a Minimax Criterion*, IEEE Trans. Computers 34 (1985), pp. 197–203.

[37] K.G. Shin and M.S. Chen, *On the number of acceptable task assignments in distributed computing systems*, IEEE Transactions on Computers 39 (1990), pp. 99–110.

[38] N.G. Shivaratri, P. Krueger, and M. Singhal, *Load Distributing for Locally Distributed Systems*, Computer 25 (1992), pp. 33–44.

[39] O. Sinnen, *Task Scheduling for Parallel Systems*, Wiley-Interscience, Hoboken, NJ (2007).

[40] R.P. Stanley, *Enumerative Combinatorics, Vol. 2*, Cambridge University Press, Cambridge, UK (1997).

[41] F. Suter, F. Desprez, and H. Casanova, *From Heterogeneous Task Scheduling to Heterogeneous Mixed Parallel Scheduling*, in *Euro-Par 2004 Parallel Processing*, M. Danelutto, M. Vanneschi, and D. Laforenza, eds., LNCS 3149, Springer, Pisa, Italy, 2004, pp. 230–237.

[42] A.S. Tanenbaum, *Distributed Operating Systems*, Prentice Hall, Upper Saddle River, NJ (1995).

[43] F.G. Tinetti, A.A. Quijano, and A.D. Giusti, *Heterogeneous Networks of Workstations and SPMD Scientific Computing*, in *1999 International Workshops on Parallel Processing*, Aizu-Wakamatsu, Japan, 1999, pp. 338–342.

[44] M.F. Tompkins, *Optimization techniques for task allocation and scheduling in distributed multi-agent operations*, Master's thesis, Massachusetts Institute of Technology, Cambridge, MA (2003).

[45] A.M. van Tilborg and G.M. Koob, *Foundations of Real-Time Computing: Scheduling and Resource Management*, Kluwer, Norwell, MA (1991).

[46] M. Šeda, *Mathematical Models of Flow Shop and Job Shop Scheduling Problems*, World Academy of Science, Engineering and Technology 31 (2007), pp. 122–127.

[47] L.L. Wang and W.H. Tsai, *Optimal assignment of task modules with precedence for distributed processing by graph matching and state-space search*, BIT 28 (1988), pp. 54–68.

[48] E.W. Weisstein, *CRC Concise Encyclopedia of Mathematics*, CRC Press, Boca Raton, FL (2003).

[49] T. Yamada and R. Nakano, *Genetic Algorithms in Engineering Systems*, in *IEE Control Engineering Series 55*, P.J. Fleming and A.M.S. Zalzala, eds., chap. Job-shop scheduling, The Institution of Engineering and Technology, London, UK, 1997, pp. 134–160.

[50] Y. Zhao, *Young tableaux and the representations of the symmetric group*, The Harvard College Mathematics Review 2 (2008), pp. 33–45.

[51] X. Zheng and S. Koenig, *K-Swaps: Cooperative Negotiation for Solving Task-Allocation Problems*, in *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, Pasadena, CA, 2009, pp. 373–378.

Appendix A. Young tableaux for job-shop and flow-shop scheduling

There are a wide variety of task assignment and scheduling problems, such as *job-shop* [46, 49], *flow-shop shop* [26, 46], and *resource-constrained project scheduling problems* [18]. Job-shop and flow-shop scheduling have also been discussed in a distributed system environment [3, 45]. In this appendix we briefly consider assignment objects using Young tableaux for the job-shop and flow-shop scheduling problem.

The job-shop scheduling problem [46, 49], in its classical form, can be described by a set of n jobs $\{J_i\}_{1 \le i \le n}$ that is to be processed on a set of m machines (or agents) $\{M_j\}_{1 \le j \le m}$. Each job consists of a chain of m operations and has a machine order to be processed, in which m operations of job J_i are processed on m different machines. The processing of job J_i on machine M_j is denoted by operation O_{ij}. Operation O_{ij} requires the processing time p_{ij} for the use of machine M_j in an uninterruptable manner. The scheduling problem is to find an assignment of all operations to minimize the *makespan* which is the maximum of the completion times (or finishing times) of all jobs (see [46, 49] for detailed assumptions and constraints for the job-shop scheduling problem).

Meanwhile, in the flow-shop scheduling problem [26, 46], each job consists of a chain of m operations and has the exact same machine order to be processed, in which m operations of a job are processed on m different machines. The scheduling problem here is to find the job sequences on the machines that minimize the makespan [46]. The flow-shop scheduling problem is a special case of the job-shop scheduling problem [26].

Now, consider the following machine orders of the jobs:

$J_1 : M_4 \to M_3 \to M_1 \to M_2,$
$J_2 : M_3 \to M_2 \to M_1 \to M_4,$
$J_3 : M_2 \to M_3 \to M_4 \to M_1.$

Note that the machine orders of the jobs for the classical, deterministic job-shop scheduling and flow-shop scheduling problem are given in advance, while job orders on machines may vary by different schedules. Note also that in the flow-shop scheduling problem, J_1, J_2, and J_3 have to be the same. An example of job orders on machines is as follows:

$M_1 : J_1 \to J_2 \to J_3,$
$M_2 : J_3 \to J_2 \to J_1,$
$M_3 : J_2 \to J_1 \to J_3,$
$M_4 : J_1 \to J_3 \to J_2.$

The assignment tableaux we have discussed in the previous sections cannot describe the job orders on machines. One possible way of describing the above job assignment using a generalized Young tableau is as follows:

1	2	3
3	2	1
2	1	3
1	3	2

The rows in the above generalized Young tableau describes the machines, in which the first row describes the first machine (i.e., M_1), the second row describes the second machine (i.e., M_2), and so on. The columns in the above generalized Young tableau describes the job orders, in which the first column describes (the index of) the first job in the job orders, the second column describes (the index of) the second job in the job orders, and so on. For a non-classical job-shop scheduling problem, each job may consist of a chain of the different number of operations. In this case we may need to use empty cells in a tableau to describe the different number of job orders on machines.

To define assignment objects using Young tableaux and groups in other types of scheduling problems, such as resource-constrained project scheduling problems [18], are problem-specific and have not been well-established thus far, which requires future re-

search.

Appendix B. Computational tools

In Section 5.2 we described an equivalence relation $\sim_{tr}$ for task reassignments on $X_n^\lambda = \{t_1, t_2, \ldots, t_n\}$ which is a set of standard generalized assignment tableaux $t_i (1 \leq i \leq n)$ of shape λ. The following figure describes how the GAP [17, 19, 20] computer algebra system can be used for finding the size of the equivalence class of $t \in X_n^\lambda$ with respect to the equivalence relation $\sim_{tr}$ by regarding X_n^λ as a set $\{1, 2, \ldots, n\}$ and defining the action of $G \leq \mathfrak{S}_n$ on $\{1, 2, \ldots, n\}$ simply by $gi = g(i)$ for $g \in G$.

```
 1: gap> g:=Group((1,2,3,4),(47,48,49,50));
 2: <permutation group with 2 generators>;
 3: gap> Elements(g);
 4: [ (), (47,48,49,50), (47,49)(48,50), (47,50,49,48), (1,2,3,4), (1,2,3,4)(47,48,49,50),
 5: (1,2,3,4)(47,49)(48,50), (1,2,3,4)(47,50,49,48), (1,3)(2,4), (1,3)(2,4)(47,48,49,50),
 6: (1,3)(2,4)(47,49)(48,50), (1,3)(2,4)(47,50,49,48),
 7: (1,4,3,2), (1,4,3,2)(47,48,49,50), (1,4,3,2)(47,49)(48,50), (1,4,3,2)(47,50,49,48) ]
 8: gap> Order(g);
 9: 16
10: gap> orbits:=Orbits(g, [1..50]);
11: [ [1, 2, 3, 4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19],
12: [20], [21], [22], [23], [24], [25], [26], [27], [28], [29],[30], [31], [32], [33], [34], [35], [36],
13: [37], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47, 48, 49, 50] ]
14: gap> OrbitLength(g,48);
15: 4
16: gap> Size(orbits);
17: 44
```

Figure 7. Examples of Propositions 5.3 and 5.4 using GAP.

Lines 1–2 in Figure 7 indicates that we choose $G \leq \mathfrak{S}_n$ that is generated by the cycles (1 2 3 4) and (47 48 49 50). Lines 3–7 in Figure 7 shows the elements of G. Lines 10–13 in Figure 7 show all the orbits when G acts on a set $\{1, 2, \ldots, 50\}$ by $gi = g(i)$ for $g \in G$. Using the renaming map $i \mapsto t_i$, we may interpret them as all the equivalence classes of $X_{50}^\lambda = \{t_1, t_2, \ldots, t_{50}\}$ with respect to $\sim_{tr}$ when G acts on X_{50}^λ by $gt_i = t_{g(i)}$ for $g \in G$. For instance, [1, 2, 3, 4] in line 11 can be interpreted as $[t_1, t_2, t_3, t_4]$. Lines 14–15 in Figure 7 indicates that the size of the orbit of $48 \in \{1, 2, \ldots, 50\}$ is 4. We may interpret it as the size of the equivalence class of $t_{48} \in X_n^\lambda$ is 4 with respect to $\sim_{tr}$. Finally, lines 16–17 show that the number of orbits is 44. We may interpret it as the number of equivalence classes of X_{50}^λ with respect to $\sim_{tr}$ is 44.

We also provide a simple tool written in GNU C++[14] for the n-task-n-agent assignment problem using *genetic algorithms* [34] and Young tableaux. Metaheuristic methods, such as genetic algorithms, are often used for task assignment problems, since finding an optimal task assignment is often intractable [11, 39]. The purpose of our tool[1], called simple tableaux-based genetic algorithms (STG), is to show how Young tableaux and their partitions are applied to the existing genetic algorithms for task assignment problems. The following table shows an example of (file) input and output of STG for the n-task-n-agent assignment problem for $n = 20$.

---

[1]Source codes and sample data are available at http://www.airesearch.kr/downloads/STG.zip.

Table 2. An example of (file) input and output of STG for the n-task-n-agent-assignment problem for $n = 20$.

1: **Input:**
2: NumberOfTasks 20
3: NumberOfAgents 20
4: $\cdots$
5: Task_Length 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200
6: Processing_Capacity 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200
7: $\cdots$
8: #GAs
9: CrossoverRate 0.8
10: MutationRate 0.005
11: $\cdots$
12: #Young Tableau : Y, Young Tabloid : N
13: YoungTableau Y
14: NumberOfPartitions 4
15: TableauShape 6 6 4 4
16: $\cdots$
17: 1
18: 2
19: $\cdots$
20: 19   16   17
21: 20   18   19

22: **Output:**
23: (GENERATION 800)
24: ********************* Candidate Solution********************
25: Candidate Solution Fitness:0.205212
26: 1 2 3 4 5 6 7 9 8 11 12 13 14 16 10 17 18 15 19 20
27: $\cdots$
28: Tableau Shape: 6 6 4 4
29: $\cdots$
30: $\{\{1, 2, 3, 4, 5, 6\}, \{7, 9, 8, 11, 12, 13\}, \{14, 16, 10, 17\}, \{18, 15, 19, 20\}\}$,
31: $\cdots$

The input of STG specifies the number of tasks, number of agents, the list of task lengths, the list of processing capacities of agents, parameters of genetic algorithms, etc. The first entry of the task length list (see line 5) in Table 2 corresponds to the task length of task ID 1, the second entry corresponds to the task length of task ID 2, and so on. Similarly, the first entry of the processing capacity list (see line 6) in Table 2 corresponds to the processing capacity of agent ID 1, the second entry corresponds to the processing capacity of agent ID 2, and so on. At the bottom of "Input" in Table 2 (see lines 17–21), the leftmost numbers denote task IDs (i.e., task IDs 1, 2, ..., 19, and 20), while their following numbers denote their precedence lists. For instance, task IDs 1 and 2 have no precedence list, while task ID 19 (respectively, task ID 20) has its precedence list [16, 17] (respectively, [18, 19]).

The output of STG shows the candidate solution represented by the one-line permutation notation for the n-task-n-agent assignment problem after a certain number of generations (see line 26 in Table 2). Then, the candidate solution represented by the one-line permutation notation is converted into the corresponding standard assignment tableau based on the given tableau shape. Finally, the converted candidate solution $\{\{1, 2, 3, 4, 5, 6\}, \{7, 9, 8, 11, 12, 13\}, \{14, 16, 10, 17\}, \{18, 15, 19, 20\}\}$ (see line 30 in Table 2) represents the standard assignment tableau of shape $(6, 6, 4, 4) \vdash 20$, where the

entries of its first row (reading from left to right) are 1, 2, 3, 4, 5, 6; the entries of its second row 7, 9, 8, 11, 12, 13; the entries of its third row 14, 16, 10, 17; and the entries of its last row are 18, 15, 19, 20.