ABACUS A Journal of Accounting, Finance and Business Studies

ABACUS, Vol. 56, No. 2, 2020 doi: 10.1111/abac.12190

ROSEMARY KIM, JAGDISH GANGOLLY, S. S. RAVI, AND DANIEL J. ROSENKRANTZ

Formal Analysis of Segregation of Duties (SoD) in Accounting: A Computational Approach

This study examines a computational framework for segregation of duties (SoD) in the design as well as implementation of accounting systems. The framework consists of a model of workflows in accounting systems based on workflow graphs, a partial order model of roles performed by the actors in the accounting system, and a specification of SoD rules. We develop a set of algorithms for four SoD rules that can be used in the enforcement of SoD. For the SoD rule that precludes task type conflicts, our results show that while compliance verification can be carried out efficiently, finding an SoD compliant assignment of tasks is computationally intractable. For those situations, we present an integer linear programming (ILP) formulation for finding compliant assignments using public domain ILP solvers. For the remaining three SoD rules, we demonstrate efficient ways of testing compliance for a given assignment as well as finding compliant assignments.

Key words: Dynamic and static controls; Internal controls analytics; Segregation of duties (SoD); SoD rules; Validity and verification of SoD.

Internal control, a centre-piece of the Sarbanes-Oxley (SOX) Act, requires that audit reports describe the scope of testing for internal controls and disclose material weaknesses in such controls. Whalen and McKeon's (2017) 13 year review of SOX 404 disclosures showed that ineffective controls continue to be an issue for non-accelerated[1] management-only[2] filers. In the fiscal year ending 2016, ineffective internal controls were reported by 1,234 (36.9%) of management-only filers, of whom 860 filers (70%) cited segregation of duties (SoD) weaknesses as

ROSEMARY KIM is with the Department of Accounting, Loyola Marymount University. JAGDISH GANGOLLY (jgangolly@albany.edu) is with the Department of Informatics, University at Albany – State University of New York. S. S. RAVI is with the Biocomplexity Institute and Initiative, University of Virginia and Department of Computer Science, University at Albany – State University of New York. DANIEL J. ROSENKRANTZ is with the Biocomplexity Institute and Initiative, University of Virginia and Department of Computer Science, University at Albany – State University of New York.

This work has been supported by a research grant from the College of Business Administration at Loyola Marymount University and partially supported by the following research grants: NSF Grant IIS-1908530, NSF Grant OAC-1916805, NSF DIBBS Grant ACI-1443054, NSF BIG DATA Grant IIS-1633028, and NSF EAGER Grant CMMI-1745207.

[1] Non-accelerated filers are public companies that have a public float of less than $75 million.

[2] Assessment by management rather than by auditors as part of SOX 404 disclosure.

one of the contributing causes. SoD therefore continues to be an important internal control focus in corporate governance, especially for non-accelerated filers as they compete in the market place with accelerated filers[3] and large accelerated filers.[4]

Much of the literature in accounting deals with internal controls in general. Examples include studies that examine the relationship between internal control weaknesses and firm size (Gao *et al*., 2009; Kinney, Jr. and Shepardson, 2011), influence of formal internal controls on norms of behavior (Tayler and Bloomfield, 2011), relationship between internal control deficiencies and firm risk and cost of equity (Ogneva *et al*., 2007; Ashbaugh-Skaife *et al*., 2009; Gordon and Wilford, 2012), relationship between internal control weaknesses and tax avoidance (Bauer, 2016), relationship between internal control weaknesses and CFO compensation (Hoitash *et al*., 2012), relationship between internal controls over financial reporting and safeguarding of corporate resources (Gao and Jia, 2016), and quality of accrual (Ashbaugh-Skaife *et al*., 2008). In some of these studies, SoD was one of the variables in understanding internal control weaknesses but not the central theme. On the other hand, there are some studies that address SoD directly such as Kobelsky (2014) and Elsas (2008). Since a large percentage of management filers state SoD issues as one of the contributing causes of internal control weaknesses, study of SoD on its own assumes importance. Our paper fills this need.

Since SoD seeks to prevent employee fraud by reducing the possibility of collusion wherever there are conflicts of interest, it has always been important for stewardship in accounting. It also has been an important concept in collateral fields such as economics, organization theory, constitutional and political theory, and even computer science. In economics, asymmetry of roles between the principal and the agents in hierarchical organizations may give rise to conflicts of interest since the agents have custody of assets, as well as powers to authorize and execute transactions and to maintain records relating to those assets. Such conflicts of interest can provide incentives to the agents to collude in order to perpetrate employee fraud on the principal (Holmstrom and Milgrom, 1991; Itoh, 1991). Segregation of duties in such situations is necessary to mitigate the incentives that exist for collusion to perpetrate employee fraud (Tirole, 1986). SoD has also occupied an important position in organization theory where increased size and complexity of tasks can cause hierarchies to evolve and vertical segregation of duties to develop so that the tasks can be coordinated and controlled (Lægaard, 2006). In constitutional law and political theory, SoD underlies the principle of separation of powers between legislative, executive, and judicial branches (Dworkin, 1978). The motivation for studying SoD in all of these fields has been to divide-and-conquer the motivations for collusion that can lead to adverse consequences.

[3] Accelerated filers are public companies that have a public float of at least $75 million and less than $700 million.

[4] Large accelerated filers are public companies that a have public float of $700 million or more.

In computer science, SoD occupies an important place in various areas including development of secure operating systems (Clark and Wilson, 1987), role-based access controls in databases (Sandhu, 1988, 1990), separation of concerns in object-oriented modeling (Aksit, 1996), layering of Internet protocols (Zimmermann, 1980), and the design of web languages (Van Kesteren and Stachowiak, 2015). Computer operating systems and accounting systems share a common objective: to monitor and control the resources they are assigned. A computer's operating system continuously gathers data on the operations being carried out and executes decisions or informs users based on established algorithms to optimize the utilization of hardware resources. An accounting system performs essentially the same functions for business operations. Therefore, an accounting system is a good metaphor for a computer operating system. A seminal paper where SoD was a shared interest in both computer science and accounting is Clark and Wilson (1987) where concepts from both areas were utilized to design multi-level security in operating systems.

Our objective in this paper is to bring to bear a computational perspective on SoD analysis in accounting systems. We accomplish this by formulating SoD problems in a rigorous mathematical framework and developing algorithms that can be used in practice for testing SoD compliance. The algorithms we develop can be used in the design as well as implementation phases of accounting systems. In the design phase the algorithms can be used to *find* assignments of tasks-to-roles and tasks-to-persons that are SoD compliant. If a compliant assignment does not exist, the accounting workflow must be redesigned or persons with different skills need to be recruited. In the implementation phase the algorithms can be used to *test* if any given assignment is SoD compliant. In transaction processing, an assignment can be used only if it is SoD compliant.

The algorithms we develop are useful to test static as well as dynamic[5] SoD properties. Static verification deals with compliance of assignments during the system design phase. Dynamic verification deals with the roles assigned during processing of transactions in real-time; it can detect deviations[6] from the default assignments and suspend transaction processing when necessary so that appropriate actions can be taken to correct the effects of the infraction. Dynamic verification therefore provides preventive and detective, as well as corrective, SoD controls over transaction processing. Our use of the terms static and dynamic is appropriate in accounting and auditing since auditors use static verification in arriving at preliminary judgements about risk relating to material weaknesses and dynamic verification during substantive testing of transactions.

Most of the research on SoD in accounting and auditing limits its focus to conflicts of interest due to the nature of tasks, and places minimal emphasis on

[5] Role-based access control literature uses static SoD to mean restrictions on roles at the time of assignment and dynamic SoD to mean restrictions on activation and availability of privileges and their duration when transactions are processed (Botha and Eloff, 2001).

[6] Deviations can arise due to contingencies such as absences, vacations, and staff changes. Such departures from design in the processing of transactions can violate SoD rules and increase the risk of employee fraud. Therefore, it is necessary to develop algorithms to test that such assignment changes do not violate SoD rules.

hierarchies and roles in organizations that can lead to such conflicts. It also abstracts away the impact of the structural properties of business processes on SoD. The accounting literature also does not address the development of algorithms for finding compliant task-role assignment to persons and for testing compliance of any assignments. Our contributions in this paper can be summarized as follows. First, we propose a graph theoretic study of SoD where accounting systems are represented by directed acyclic graphs constructed from accounting workflows, and privileges are bundled into roles which are hierarchically organized. Second, we exploit the structural properties of accounting workflows to propose SoD rules that incorporate conflict of interest (CoI) arising out of causes other than the nature of tasks. Such rules considerably reduce the size of the SoD rulebase. Third, we develop algorithms for finding as well as testing compliant assignments that can be used in static as well as dynamic contexts. Fourth, we study the efficiency of each algorithm, and where the underlying problem is computationally intractable, we suggest a practical solution approach using integer linear programming.

THE SETTING

Our study of SoD in accounting systems is based on a foundation consisting of three parts: a model of workflow, a set of roles that are hierarchically organized, and a set of SoD rules. We describe workflows in accounting systems by identifying tasks and specifying preconditions that must be satisfied for the task to be initiated. We consider a set of roles where each role is a bundle of privileges to access information in the accounting system. The set of roles is assumed to be given and organized hierarchically. The SoD rules specify how tasks in the workflow and the roles should be assigned to persons such that segregation of duties is maintained. Given a set of persons, an important function of accounting system design is to assign tasks and roles to persons in such a way that SoD rules are complied with. Those assignments also must adhere to the constraints imposed by the skill requirements for the performance of tasks and skills possessed by the persons. This model allows us to examine whether the design of an accounting system and the processing of transactions comply with given SoD rules, if compliant allocation of tasks to persons can be computed when they exist, and whether the algorithms are efficient.[7]

Overview

In this section we provide definitions of terms used in this paper, such as tasks, preconditions, persons, privileges, roles, role sets, assignments, and validity of task-roles to people. Many of these concepts are integral to the two models commonly used in business processes: model of accounting workflows using the

[7] The efficiency of an algorithm is estimated by its running time as a function of the size of input data, and is expressed using the Big O notation. An algorithm is efficient if that function is a polynomial (Cormen *et al.*, 2009).

business process modelling notation (BPMN), and model of partial order of roles. Our work on SoD is built on these two models. We illustrate these concepts using an example of a purchasing system in accounting. Next, we provide a short summary of matrix representations of the data in SoD problems and the graph theoretic definitions needed for the rest of the paper. The final subsection presents efficient algorithms for determining whether a given assignment is valid and for determining whether there is a valid assignment for a given business process, when no SoD rules are specified. These algorithms are useful in developing algorithms for the corresponding problems when an SoD rule is also specified.

Basic Definitions

Model of workflows Accounting workflows monitor business processes, and they are commonly modelled using the BPMN (Juhás *et al*., 2009; Best and Desel, 1990; Deelman *et al*., 2005; Sadiq and Orlowska, 1999; Wu *et al*., 2013; Deelman *et al*., 2015). The model of workflows consists of a directed acyclic graph (dag) D (V, A) where each node in V is a task, and a directed edge $(u, v) \in A$ indicates that the task corresponding to u must be completed before starting the one corresponding to v. We refer to D as the *precedence dag* of the business process. Each node in this graph corresponds to a task in the model of the business process for which SoD is being considered.

Privileges, roles, and role hierarchy Performing tasks requires access to information called *privileges*.[8] In general, a task requires more than one privilege. Let $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ be the set of privileges. While individual privileges can be assigned directly to persons, it is more efficient to first bundle privileges into a set of **roles** $R = \{r_1, r_2, \ldots, r_m\}$ so that each role is a subset of Π; that is, for any $r_i \in R$, $r_i \subseteq \Pi$. We assume that the set R of roles is given. With n privileges in Π, there are 2^n possible roles. However, in most systems the number of roles actually used will be much smaller.

Since roles in R are defined as subsets of the set of privileges Π, there is a natural partial order $<_R$ on the set of roles R. Given any two roles r_i and r_j, the ordered pair $(r_i, r_j) \in <_R$ if the set of privileges associated with r_i is a *strict* subset of the set of privileges associated with r_j. If $(r_i, r_j) \in <_R$, we say that r_j **strictly dominates** r_i. Two roles r_i and r_j are **incomparable** if neither (r_i, r_j) nor (r_j, r_i) appears in $<_R$. We assume that $<_R$ is *irreflexive*; that is, for each $r \in R$, $(r,r) \notin <_R$. (Thus, no role strictly dominates itself.) This partial order reflects the organizational hierarchy of privileges.

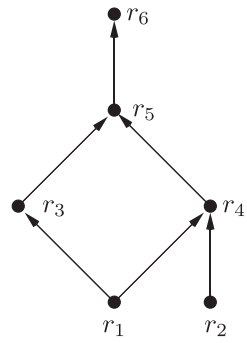
We now present an example to illustrate these concepts. Let $\Pi = \{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6\}$ be a set of six privileges. Let roles r_1, r_2, r_3, r_4, r_5, and r_6 be defined as subsets of Π as shown in the table of Figure 1. The domination

[8] Most modern database management systems are implemented using privileges. The privileges to access data are bundled into *roles,* and the privileges to access system resources such as terminals and computers are bundled into *profiles*. Default roles and profiles are assigned to people in the organization, but they may be reassigned for each session used for processing a transaction.

FIGURE 1

AN EXAMPLE TO ILLUSTRATE ROLE HIERARCHY

Role	Privileges
r_1	$\{\pi_1\}$
r_2	$\{\pi_2\}$
r_3	$\{\pi_1, \pi_3\}$
r_4	$\{\pi_1, \pi_2, \pi_4\}$
r_5	$\{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5\}$
r_6	$\{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6\}$



Note: For simplicity, domination relationships implied by transitivity are not shown in the directed graph.

relationship $<_R$ among these roles is shown as a directed graph[9] in the figure; this graph has a directed edge from r_i to r_j when r_j strictly dominates r_i (that is, the pair $(r_i, r_j) \in <_R$). For example, r_3 strictly dominates r_1 since the set of privileges associated with r_1 (namely, $\{\pi_1\}$) is a strict subset of that associated with r_3 (namely, $\{\pi_1, \pi_3\}$). Similarly, r_4 strictly dominates r_2. Roles r_2 (privilege set = $\{\pi_2\}$) and r_3 (privilege set = $\{\pi_1, \pi_3\}$) are incomparable. To avoid clutter, in the directed graph of Figure 1, we have not shown the domination edges which are implied by transitivity. For example, since r_5 strictly dominates r_3 and r_3 strictly dominates r_1, it follows by transitivity that r_5 strictly dominates r_1. However, the directed edge (r_1, r_5) is not shown in the figure.

Tasks, role sets, and persons Consider an organization with a set T of tasks (given in the model of workflows above) and a set P of persons. For each task, there is a set of roles required to complete it. This is captured by a function f_{TR}: $T \rightarrow 2^R$ which specifies for each task $t \in T$, the subset of roles needed to complete t. These roles may be assigned to one or more persons. If all the required roles are assigned to one person, that person is solely responsible for the performance of the task. On the other hand, if the required roles are assigned to more than one person, those persons are jointly responsible for the performance of the task. This enables one to divide the responsibility for a complex task among two or more persons in order to ensure that collusion is required to perpetrate employee fraud. That is one of the fundamental principles of SoD. We will refer to $f_{TR}(t)$ as the **role set** of task t.

Similarly, there is a set of roles that can be assigned to a person, and that set is determined by the skills possessed by that person. This is captured by a function

[9] This directed graph represents the Hasse Diagram (Liu, 1985) of the partial order.

f_{PR}: $P \rightarrow 2^R$ which specifies for each person $p \in P$, the subset of roles that can be assigned to p. We will refer to $f_{PR}(p)$ as the **role set** of person p.

Business process Sets P, T, and R, along with functions f_{TR} and f_{PR}, constitute a **business process** $\mathbb{P}$; that is,

$$\mathbb{P} = \{P, T, R, f_{TR}, f_{PR}\} \tag{1}$$

The tasks are represented in the dag for the business process whereas P, R, and the partial order $<_R$ on the set of roles form the organizational model. Functions f_{TR} and f_{PR} interface the business process model and the organizational model in the implementation of SoD.

Task type For each task $t \in T$, its **task type**, denoted by $\lambda(t)$, is given. In this paper, we consider four common task types, namely `A` (authorization), `C` (custody), `E` (execution), and `R` (recording). Thus, for each $t \in T$, $\lambda(t) \in$ {`A`, `C`, `E`, `R`}. Auditors often categorize tasks into the above task types to study SoD. Our SoD rules are based on these four task types. However, our algorithms are general and they do not rely on the number of roles being a small integer. In presenting our analysis results, we will use τ as the number of task types.

Assignment Tasks must be assigned to persons. This can be accomplished by first determining the roles required for performing the tasks and then assigning the task-role pairs to persons. For this, we define the binary relation $\mathbb{B}$ from T to R (that is, $\mathbb{B} \subseteq T \times R$) as follows:

$$\mathbb{B} = \{(t,r) : t \in T \text{ and } r \in f_{TR}(t)\} \tag{2}$$

Thus, for each $t \in T$ and each role r in the role set of t, the set $\mathbb{B}$ contains the pair (t, r). Note that $\mathbb{B}$ can be computed from sets T, R, and the function f_{TR}.

An **assignment** α is a function from $\mathbb{B}$ to P. For each task-role pair $(t,r) \in \mathbb{B}$, $\alpha(t,r)$ specifies the person p who is assigned that task-role pair. Throughout this paper, we use the phrase 'a person is assigned a task' to mean that the person is assigned one or more roles in that task.

Validity of assignments An assignment α is **valid**[10] if for every $(t,r) \in \mathbb{B}$, the following condition holds: if $\alpha(t, r) = p$ then $r \in f_{PR}(p)$. Thus, a valid assignment has the property that if a person p is assigned role r in task t, then r is in the role set of p. This ensures that no person p is assigned to a task that p cannot handle.

We also encapsulate information relating to business process concepts in three matrices called TR (task-role), RP (role-person), and ALPHA (assignment) as discussed in the next section and summarized in Appendix A. They will be used in deriving some of our results. In Appendix A we summarize concepts from graph theory that are used in our algorithms and proofs.

[10] Validity is sometimes referred to in the literature as *soundness* (Knorr and Weidner, 2001).

PURCHASE PROCESS EXAMPLE

Overview

We consider an example of a purchase process to illustrate some of our analytical results relating to validity and viability[11] of assignments for four SoD rules. Since SoD Rule 1 is used extensively in audit practice, we use this purchase process example to illustrate Algorithm 1 to test the *validity* of a given assignment, and Algorithm 3 to test the *viability* of a given assignment. For the remaining three SoD rules we construct simpler examples.[12] We also establish results on *finding* viable assignments if they exist. However, detailed examples are not provided as finding viable assignments are beyond the scope of accounting and more relevant to information system design.

Model of Workflows for the Purchase Process

Consider a purchase process $\mathbb{P}$ consisting of tasks $T = \{t_1, t_2, \ldots, t_{11}\}$ and preconditions consisting of arcs $A = \{a_1, a_2, \ldots, a_{12}\}$. Figure 2 shows the dag for the process $\mathbb{P}$ where circles represent tasks t_i and arcs a_i represent preconditions for the tasks. For example, arc a_1 shows that task t_1 must be completed before task t_2 can be started, and arcs a_3 and a_4 collectively show that both tasks t_2 and t_3 must be completed before task t_4 can be started. For $\mathbb{P}$ to be completed each of the tasks in T must be completed.

Privileges and Roles for the Purchase Process Example

For each task in the business process $\mathbb{P}$ represented by the dag in Figure 2, Table 1 provides tasks, their description, role sets, and privileges needed to accomplish the tasks. The first column of Table 1 gives the task number. The second column specifies the role set of the task. For example, to perform task t_6 ('Prepare RR'), role r_6 is required. The third column provides a description of the task, and the type of the task is shown within parentheses. The last column gives the privileges required to perform the task. For example, task 'Distribute PR' (t_1) requires that the person performing that task must play role r_1, and task t_6 requires that the person performing it must play role r_6.

TR Matrix

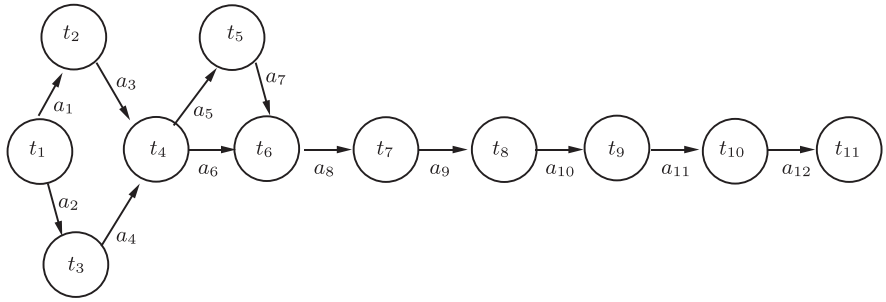
The information on the role sets of tasks can be summarized in a matrix TR as in Table 2. In this matrix, each row represents a task and each column represents a role. An entry TR[i, j] is 1 if task i requires role j, and 0 otherwise. For example, the role 'Privilege to access purchase requisition' (r_1) is required for the task 'Distribute purchase requisition' (t_1), and so the element TR[1, 1] in the matrix TR has the value 1. Since t_1 does not require any other role, all remaining entries in the first row of matrix TR are 0. Also, since the role set of task 'Prepare receiving report' (t_6) has r_6, in the sixth row of the TR matrix, the entry in the sixth column is 1. Since t_6 does not

[11] An assignment is valid if the role assigned to every person is in their role set. Further, an assignment is viable under an SoD rule if it is valid and compliant with the said rule.

[12] Constructing accounting examples for those rules would have necessitated use of very large business processes that would not illustrate well the working of our algorithms.

FIGURE 2

A DIRECTED ACYCLIC GRAPH OF A PURCHASE PROCESS $\mathbb{P}$



t_1:	Distribute PR	t_2:	Review & approve PR for vendor & price
t_3:	Review & approve PR for budget	t_4:	Distribute approved PR/PO
t_5:	Send PO to vendor	t_6:	Prepare RR
t_7:	Update inventory	t_8:	Match PO, RR, and VI
t_9:	Approve payment of vendor invoice	t_{10}:	Prepare and sign check
t_{11}:	Mail check to vendor		

require any other role, all other entries in the sixth row of TR are 0. In general, a task may need more than one role, and a role may be needed by more than one task.

Role Sets of Persons and the RP Matrix

In this example, there are 12 persons labelled p_1, p_2, p_3, …, p_{12}. The roles that persons can play are determined by their skills and captured by the function f_{PR}: $P \rightarrow 2^R$. We assume that the function f_{PR} is specified by Table 3.

The information in Table 3 is expressed in the matrix RP in Table 4. In this matrix, rows represent the roles and columns represent the persons. An element of the RP matrix has value 1 if the role corresponding to the row can be played by the person corresponding to the column. For example, in the matrix in Table 4, the value of element RP[7, 12] is 1 because the role r_7 can be played by person p_{12}. But the value of element RP[7, 4] is 0 because role r_7 cannot be played by person p_4.

Task-role Assignments

We will consider the assignment of persons to tasks summarized in Table 5 for the discussion of the workings of the algorithms.

The Matrix ALPHA

The assignment in the above section can be summarized in the matrix ALPHA where the rows denote tasks and columns denote the roles. Thus, the rows and columns of ALPHA have the same denotations as the TR matrix. However,

TABLE 1

PRIVILEGES, TASK DESCRIPTIONS, AND ROLE SETS OF TASKS

Task #	Role set of task (f_{TR})	Task Description	Privileges
t_1	r_1	Distribute PR (**E**)	π_1: Privilege to access PR
t_2	r_2	Review & approve PR for vendor & price (**A**)	π_2: Privilege to access to approved vendor list π_3: Privilege to access price list
t_3	r_3	Review & approve PR for budget (**A**)	π_4: Access to budget information π_5: Privilege to approve PR
t_4	r_4	Distribute approved PR/PO (**E**)	π_6: Privilege to access PO π_7: Privilege to distribute PO
t_5	r_5	Send PO to vendor (**E**)	π_6: Privilege to access PO π_8: Privilege to mail PO to vendors
t_6	r_6	Prepare RR (**C**)	π_9: Privilege to receive & open the package π_6: Privilege of access to PO
t_7	r_7	Update inventory (**R**)	π_{10}: Privilege to access RR π_{11}: Privilege to input inventory received
t_8	r_8	Match PO, RR, and VI (**R**)	π_6: Privilege to access PO π_{12}: Privilege to access RR π_{13}: Privilege to access VI π_{14}: Privilege to update the match results
t_9	r_9	Approve payment of vendor invoice (**A**)	π_2: Privilege to access to approved vendor list π_3: Privilege to access price list π_{15}: Privilege to authorize payments to vendors
t_{10}	r_{10}	Prepare and sign check (**A**)	π_{16}: Privilege to sign the check
t_{11}	r_{11}	Mail check to vendor (**E**)	π_{17}: Privilege to mail check

A: Authorization **C**: Custody **E**: Execution **R**: Recording PR: Purchase Requisition PO: Purchase Order RR: Receiving Report BOL: Bill of Lading VI: Vendor Invoice

TABLE 2

MATRIX FOR ROLE SETS OF TASKS (TR)

	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}
t_1	1	0	0	0	0	0	0	0	0	0	0
t_2	0	1	0	0	0	0	0	0	0	0	0
t_3	0	0	1	0	0	0	0	0	0	0	0
t_4	0	0	0	1	0	0	0	0	0	0	0
t_5	0	0	0	0	1	0	0	0	0	0	0
t_6	0	0	0	0	0	1	0	0	0	0	0
t_7	0	0	0	0	0	0	1	0	0	0	0
t_8	0	0	0	0	0	0	0	1	0	0	0
t_9	0	0	0	0	0	0	0	0	1	0	0
t_{10}	0	0	0	0	0	0	0	0	0	1	0
t_{11}	0	0	0	0	0	0	0	0	0	0	1

TABLE 3

ROLE SETS OF PERSONS

Person	Role Set of Person	Person	Role Set of Person
p_1	$\{r_1, r_4, r_{11}\}$	p_7	$\{r_7, r_8\}$
p_2	$\{r_2, r_3, r_9\}$	p_8	$\{r_7, r_8\}$
p_3	$\{r_2, r_3, r_{11}\}$	p_9	$\{r_8, r_9\}$
p_4	$\{r_1, r_4, r_{11}\}$	p_{10}	$\{r_9, r_{10}\}$
p_5	$\{r_5, r_{11}\}$	p_{11}	$\{r_1, r_5, r_{11}\}$
p_6	$\{r_6, r_7, r_8\}$	p_{12}	$\{r_3, r_7, r_{10}\}$

TABLE 4

MATRIX FOR ROLE SETS OF PERSONS (RP)

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}
r_1	1	0	0	1	0	0	0	0	0	0	1	0
r_2	0	1	1	0	0	0	0	0	0	0	0	0
r_3	0	1	1	0	0	0	0	0	0	0	0	1
r_4	1	0	0	1	0	0	0	0	0	0	0	0
r_5	0	0	0	0	1	0	0	0	0	0	1	0
r_6	0	0	0	0	0	1	0	0	0	0	0	0
r_7	0	0	0	0	0	1	1	1	0	0	0	1
r_8	0	0	0	0	0	1	1	1	1	0	0	0
r_9	0	1	0	0	0	0	0	0	1	1	0	0
r_{10}	0	0	0	0	0	0	0	0	0	1	0	1
r_{11}	1	0	1	1	1	0	0	0	0	0	1	0

TABLE 5

TASKS, ROLES, AND ASSIGNMENTS

Task	Role	Assignment
t_1	r_1	$\alpha(t_1, r_1) = p_4$
t_2	r_2	$\alpha(t_2, r_2) = p_3$
t_3	r_3	$\alpha(t_3, r_3) = p_2$
t_4	r_4	$\alpha(t_4, r_4) = p_1$
t_5	r_5	$\alpha(t_5, r_5) = p_{11}$
t_6	r_6	$\alpha(t_6, r_6) = p_6$
t_7	r_7	$\alpha(t_7, r_7) = p_7$
t_8	r_8	$\alpha(t_8, r_8) = p_9$
t_9	r_9	$\alpha(t_9, r_9) = p_{10}$
t_{10}	r_{10}	$\alpha(t_{10}, r_{10}) = p_{12}$
t_{11}	r_{11}	$\alpha(t_{11}, r_{11}) = p_5$

while the entries in the TR matrix are zeroes and ones, those in the ALPHA matrix denote the identity of the persons (labels) to whom the task-role combinations are assigned. The ALPHA matrix for the assignments in Table 5 is given in Table 6.

TABLE 6

MATRIX ALPHA FOR THE ASSIGNMENT SHOWN IN TABLE 5

	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}
t_1	p_4	0	0	0	0	0	0	0	0	0	0
t_2	0	p_3	0	0	0	0	0	0	0	0	0
t_3	0	0	p_2	0	0	0	0	0	0	0	0
t_4	0	0	0	p_1	0	0	0	0	0	0	0
t_5	0	0	0	0	p_{11}	0	0	0	0	0	0
t_6	0	0	0	0	0	p_6	0	0	0	0	0
t_7	0	0	0	0	0	0	p_7	0	0	0	0
t_8	0	0	0	0	0	0	0	p_9	0	0	0
t_9	0	0	0	0	0	0	0	0	p_{10}	0	0
t_{10}	0	0	0	0	0	0	0	0	0	p_{12}	0
t_{11}	0	0	0	0	0	0	0	0	0	0	p_5

Binary Relation $\mathbb{B}$ *from* T *to* R

The binary relation $\mathbb{B}$, which includes all task-role pairs (t, r) such that the role r is in the role set of the task t, is another representation of the TR matrix. This binary relation is useful because if the TR matrix is sparse (that is, contains only small number of entries which are 1), the memory space needed to store $\mathbb{B}$ will be significantly smaller than that used by the TR matrix. For the TR matrix shown in Table 2, the binary relation $\mathbb{B}$ is as follows:

$$\mathbb{B} = \{(t_1,r_1),(t_2,r_2),(t_3,r_3),(t_4,r_4),(t_5,r_5),(t_6,r_6),(t_7,r_7),(t_8,r_8),(t_9,r_9),(t_{10},r_{10}),(t_{11},r_{11})\} \quad (3)$$

PRELIMINARY RESULTS WITHOUT SoD RULES

Given an assignment, we should be able to determine if it is valid. We should also be able to determine if a business process $\mathbb{B}$ has a valid assignment; if there is no such assignment, one can conclude that the persons in P do not possess all the skills required for processing transactions. In this section we present our preliminary results regarding valid assignments assuming that the matrices TR and RP and the assignment function α given as the matrix ALPHA in the above section are available. The algorithms given in this subsection do not consider any SoD rules. In the subsequent sections of this paper, we present algorithms for testing the validity of a given assignment and determining whether there is a valid assignment when an SoD rule is also specified.

Lemma 1: *There are polynomial time algorithms for the following problems: (i) determining whether a given assignment α is valid and (ii) determining whether there is a valid assignment for a given business process.*

Proof of Part (i): A simple procedure to test whether a given assignment α is valid is shown in Algorithm 1.

Algorithm 1. Testing whether a given assignment α is valid (without any SoD rule)

1 for *each pair* $(t,r) \in \mathbb{B}$ **do**

2 Let $p = \alpha(t, r)$.

3 Check whether r is in the role set of p; if not, **output** "Assignment α is invalid" and **stop**.

4 end

5 output "Assignment α is valid".

The correctness of the algorithm is obvious since it directly applies the definition of validity for the given assignment α. To estimate the running time, note that the number of iterations of the loop in Step 1 is at most $|\mathbb{B}|$. Since the function α is specified by the ALPHA matrix, for given values of t and r, the value $\alpha(t, r)$ can be obtained in $O(1)$ time. Further, we can determine whether r is in the role set of person p in $O(1)$ time by checking whether the entry $RP[r, p] = 1$. Thus, the algorithm runs in $O(|\mathbb{B}|)$ time.

EXAMPLE 1

We provide an example to illustrate the working of Algorithm 1 based on the running example described in the preceding section. The data is summarized in the matrices TR and RP shown in Tables 2 and 4. For each pair $(t,r) \in \mathbb{B}$, Algorithm 1 tests if role r is in the role set of the person who is assigned the task-role pair. If the role is not in the role set of the person, the assignment is not valid, and the algorithm stops. Table 7 shows the iterations in the working of the algorithm. The first two columns show the iteration number and the (t, r) pair from $\mathbb{B}$ considered in that iteration. The third column gives the person to whom the (t, r) pair in column two is assigned as given in the matrix ALPHA in Table 6. The fourth column indicates whether the assignment is valid, and the last column gives the reason why. The validity of an assignment depends on whether the role is in the role set of the person; this is determined by checking the element $RP[t, r]$. We can conclude from Table 7 that the assignment given by the matrix ALPHA is valid.

Now, suppose we swap the tasks assigned to persons p_3 and p_5 so that p_3 is assigned task t_{11} and p_5 is assigned task t_2. Table 8 shows the working of the Algorithm 1 for the modified assignment. It can be seen that the algorithm will stop after running the first two iterations because the assignment of task t_2 to person p_5 is not valid as the role r_2 required by task t_2 is not in the role set of the person p_5.

TABLE 7

WORKING OF ALGORITHM 1 WHERE THE GIVEN ASSIGNMENT IS VALID

Iter.	Pair Considered	Chosen Assignment	Validity	Reason
1	(t_1, r_1)	$\alpha(t_1, r_1) = p_4$	Valid	$RP[r_1, p_4] = 1$
2	(t_2, r_2)	$\alpha(t_2, r_2) = p_3$	Valid	$RP[r_2, p_3] = 1$
3	(t_3, r_3)	$\alpha(t_3, r_3) = p_2$	Valid	$RP[r_3, p_2] = 1$
4	(t_4, r_4)	$\alpha(t_4, r_4) = p_1$	Valid	$RP[r_4, p_1] = 1$
5	(t_5, r_5)	$\alpha(t_5, r_5) = p_{11}$	Valid	$RP[r_5, p_{11}] = 1$
6	(t_6, r_6)	$\alpha(t_6, r_6) = p_6$	Valid	$RP[r_6, p_6] = 1$
7	(t_7, r_7)	$\alpha(t_7, r_7) = p_7$	Valid	$RP[r_7, p_7] = 1$
8	(t_8, r_8)	$\alpha(t_8, r_8) = p_9$	Valid	$RP[r_8, p_9] = 1$
9	(t_9, r_9)	$\alpha(t_9, r_9) = p_{10}$	Valid	$RP[r_9, p_{10}] = 1$
10	(t_{10}, r_{10})	$\alpha(t_{10}, r_{10}) = p_{12}$	Valid	$RP[r_{10}, p_{12}] = 1$
11	(t_{11}, r_{11})	$\alpha(t_{11}, r_{11}) = p_5$	Valid	$RP[r_{11}, p_5] = 1$

Conclusion: The given assignment is valid.

TABLE 8

WORKING OF ALGORITHM 1 WHERE THE GIVEN ASSIGNMENT IS NOT VALID

Iter.	Pair Considered	Chosen Assignment	Validity	Reason
1	(t_1, r_1)	$\alpha(t_1, r_1) = p_1$	Valid	$RP[r_1, p_1] = 1$
2	(t_2, r_2)	$\alpha(t_2, r_2) = p_5$	Invalid	$RP[r_2, p_5] = 0$

Conclusion: The given assignment α is invalid.

Proof of Part (ii): A simple procedure to find a valid assignment α (when one exists) is shown in Algorithm 2.

Algorithm 2. Finding a valid assignment (if one exists) without any SoD rule

```
1 for each pair (t,r)∈𝔹 do
2     if there is a person p such that r is in the role set of p then
3         Set α(t, r) = p.
4     else
5         output "No valid assignment" and stop.
6     end
7 end
8 output the assignment α.
```

The algorithm tries to construct a valid assignment α as follows. For each $(t,r) \in \mathbb{B}$, if there is a person p who can play role r (that is, r is in the role set of p), it chooses p as the value of $\alpha(t, r)$ and tries the next pair in $\mathbb{B}$. If there is no such person, it can be seen that there is no valid assignment and the algorithm stops after producing an appropriate message. If the algorithm succeeds in finding a person for each task-role pair in $\mathbb{B}$, it outputs the constructed assignment α. Thus, the algorithm is correct.

To develop an efficient implementation of the algorithm, we first preprocess the RP matrix and construct an array $\mathbf{Z}[1 \ldots |P|]$ so that $\mathbf{Z}[r]$ stores a person p who can play role r. This can be done by going through each row r of the RP matrix (whose rows correspond to roles) and finding a column p such that $\mathrm{RP}[r, p] = 1$. (If all the entries in row r of the RP matrix are zero, then we set $\mathbf{Z}[r] = -1$ to indicate that there is no person who can play the role r.) Clearly, the time needed to construct the array $\mathbf{Z}$ is $O(|R| \times |P|)$. Given the array $\mathbf{Z}$, we can find in $O(1)$ time whether there is a person who can play a given role r. Now, to estimate the running time of Algorithm 2, note that the number of iterations of the loop in Step 1 is at most $|\mathbb{B}|$. Once the array $\mathbf{Z}$ is available, the time used in each iteration of the loop is $O(1)$. So, the overall running time is $O(|\mathbb{B}| + |R| \times |P|)$. Since $\mathbb{B}$ and the RP matrix are part of the input to the problem, the running time is linear in the input size. □

In our purchase process example, $|\mathbb{B}|$ is 11, the number of task role combinations in equation (3). The number of roles $|R|$ is 11, which is the number of rows in Table 1. The number of people $|P|$ is 12, which is the number of columns in Table 4. The running time of the algorithm is given by $O(|\mathbb{B}| + |R| \times |P|)$ equals $11 + (11 \times 12) = 143$. Since $\mathbb{B}$ and RP matrix are part of the input to the problem, the running time is linear in the input size.

We now present two examples to illustrate Algorithm 2. For brevity, we have chosen a smaller business process to illustrate this algorithm.

EXAMPLE 2

Consider a business process whose task set T is given by $T = \{t_1, t_2, t_3\}$. Let $R = \{r_1, r_2, r_3, r_4\}$ denote the set of roles needed for the tasks and let $P = \{p_1, p_2, p_3, p_4\}$ denote the persons in the organization. Further, let the task-role (TR) matrix and the role-person (RP) matrices for this business process be as shown in Figure 3.

From the TR matrix, the set $\mathbb{B}$ of task-role pairs is given by

$$\mathbb{B} = \{(t_1,r_1), (t_1,r_2), (t_2,r_2), (t_2,r_3), (t_3,r_2), (t_3,r_4)\} \tag{4}$$

We will apply Algorithm 2 to check whether there is a valid assignment α, and if so, find one such assignment. Algorithm 2 considers each pair in $\mathbb{B}$ and tries to

FIGURE 3

TR AND RP MATRICES FOR EXAMPLE 2

TR Matrix

	r_1	r_2	r_3	r_4
t_1	1	1	0	0
t_2	0	1	1	0
t_3	0	1	0	1

RP Matrix

	p_1	p_2	p_3	p_4
r_1	1	0	1	1
r_2	1	1	0	0
r_3	1	1	0	1
r_4	0	1	1	0

TABLE 9

WORKING OF ALGORITHM 2

Iter.	Pairs(t, r)	Chosen Assignment	Validity	Reason
1	(t_1, r_1)	$\alpha(t_1, r_1) = p_1$	Valid	$RP[r_1, p_1] = 1$
2	(t_1, r_2)	$\alpha(t_1, r_2) = p_1$	Valid	$RP[r_2, p_1] = 1$
3	(t_2, r_2)	$\alpha(t_2, r_2) = p_1$	Valid	$RP[r_2, p_1] = 1$
4	(t_2, r_3)	$\alpha(t_2, r_3) = p_2$	Valid	$RP[r_3, p_2] = 1$
5	(t_3, r_2)	$\alpha(t_3, r_2) = p_2$	Valid	$RP[r_2, p_2] = 1$
6	(t_3, r_4)	$\alpha(t_3, r_4) = p_3$	Valid	$RP[r_4, p_3] = 1$

assign a person to that pair. Table 9 shows the assignment chosen in each of the six iterations (one corresponding to each pair in $\mathbb{B}$).

Thus, in this case, the algorithm finds a valid assignment given by the third column of the above table. We note that in an assignment resulting from the algorithm, there may be people with multiple roles (e.g., p_1 is assigned roles r_1 and r_2) and others who are not assigned any role (e.g., p_4 is not assigned any role).

EXAMPLE 3

Suppose we change the RP matrix in Example 2 above so that all the entries in the row corresponding to r_4 are zero (that is, there is no person who can play the role r_4). The first five iterations will proceed in the manner indicated in Table 9. In Iteration 6, when the algorithm considers the pair (t_3, r_4), it cannot find a person who can be assigned role r_4. Therefore, the algorithm will output the message 'No valid assignment'.

In this section we developed a rigorous way of determining the validity of assignments of task-role pairs to persons. The validity criterion ensures that the assignments are consistent with the requirements of tasks and the skills of persons.

CONFLICT OF INTEREST AND SoD RULES

Explicit and formal statements of SoD rules are essential for the development of any system in enforcing SoD policies. However, it is very difficult to come up with an exhaustive set of SoD rules that can be used in any situation. It is also challenging to gather external social network information that is not usually collected by organizations. For example, social networks such as family relationships can lead to CoI. While this issue is important in the study of SoD, this paper focuses on factors that are within the control of the organization and so beyond the scope of this paper. Therefore we exploit the structural properties of business processes and role hierarchies to derive SoD rules that are relevant and parsimonious. The data relating to these properties are readily available since they are maintained by most organizations even if only to satisfy the requirement of audits.

In accounting, there are no studies of formal SoD rules except for those that incorporate them in conflict of interest matrices whose rows and columns represent tasks. The entries are a measure of the risk used to assign tasks to persons. In computer science there is a study by Knorr and Weidner (2001) that formalizes SoD rules, but does not present rule descriptions that exploit the structure of business processes. In this section, we introduce SoD rules in the context of conflicts of interest, give their informal definitions, provide the motivation for the traditional way of studying SoD only in terms of conflicts due to the type of tasks, and finally discuss each of the four SoD rules considered in this paper. The following section provides analytical results on the efficiency of the algorithms for testing compliance of any given assignment as well as finding compliant assignments for each SoD rule.

SoD rules encapsulate policies to mitigate the effects of conflicts of interest. So, given a task and the person to whom it is assigned, an SoD rule determines if any other task can or cannot be assigned to any person in the organization (Knorr and Weidner, 2001). For example, for any two tasks t_i and t_j in a business process where task t_i precedes task t_j, a typical SoD rule can be expressed as

$$(t_i,p_k) \rightarrow \neg(t_j,p_l) \tag{5}$$

where (t_i, p_k) indicates that task t_i is assigned to person p_k. The above rule states that if task t_i is assigned to person p_k, then a subsequent task t_j cannot be assigned to person p_l. This is a generalization of the traditional accounting SoD rule for distinct tasks t_i and t_j and $k = l$. However, a disadvantage of this approach is that for business processes of any size, the number of SoD rules will be very large since, to be comprehensive, the rules must compare each pair of tasks. So, if there are n tasks, the number of SoD rules will be at least $n \times (n - 1) \approx n^2$. However, merely classifying tasks into types ignores the structural properties of the business process and the organization. The number of rules can therefore be reduced considerably by exploiting those structural properties. That is the strategy we adopt in the rest of this paper. We now motivate the four SoD rules we study in this paper.

CoI Due to Task Type

Auditors have traditionally used just the nature of tasks to classify them into four types: authorization, execution, recording, and custody. However, in general, we can have an arbitrary number of task types so long as they relate to SoD. A presumption of conflicts of interest is created when any two or more task types are assigned to the same person, and the first SoD rule we study states that *no person should be assigned a role in more than one type of task within a business process*. This can be considered the baseline SoD Rule, and is our SoD Rule 1. However, it does not consider conflicts of interest that arise due to factors other than those due to task type. Our analytical results for this SoD Rule are in the first subsection of the following section, titled 'SoD Rule 1: CoI Due to Task Type'.

Conflicts of interest can also arise due to the following three factors: role conflicts, role dominance, or common roles among tasks. In all of these three situations, conflicts arise in large volume transaction processing due to social interactions occasioned by two tasks that are part of an execution chain. An execution chain in a business process for the purposes of this paper is a sequence of tasks that are collectively important for maintaining control over the business process. Such execution chains are a structural property of business processes that can be analytically derived as *place invariants* of the Petri Net underlying the business process (Lautenbach, 1987; Girault and Valk, 2001; Yamalidou *et al*., 1996).

CoI Due to Role Conflicts

Conflicts of interest can arise due to the nature of roles that persons assume in processing transactions. Since roles are bundles of privileges, conflicts can arise because information can leak through a person whose role provides a wider set of privileges than those required to perform the task. This breaks the cardinal rule of internal control that each person should have only the minimum set of privileges required to perform a task (principle of minimum privilege). Controlling role conflicts is important because roles are constantly being activated and deactivated in such contexts, and errors in the assignment of task-role pairs to persons can increase the risk of employee fraud. One way to avoid such conflicts of interest is to require that a person assume at most one role within a business process. Therefore, our SoD Rule 2 states that *no person may perform more than one role within a business process*. Our analytical results for this SoD Rule are in the second subsection of the following section, titled 'SoD Rule 2: CoI Due to Role Conflicts'.

CoI Due to Role Dominance

In organizations that are organized hierarchically, conflicts of interests can arise when for any two tasks which are in a chain of execution in a business process and which are performed by different employees, the role assumed by one dominates the role assumed by the other. This is because the person assuming the dominant role can perpetrate fraud by coaxing the one assuming the dominated role into colluding. One way to avoid such conflicts of interest is to require that when any two persons are assigned tasks of the same type within a chain (that is, directed path), they should not be assigned roles where one role strictly dominates the other.

Therefore, our third SoD Rule 3 (Non-Dominating Roles Along Directed Paths) states the following: *any two persons who are assigned tasks of the same type within a directed path should not be assigned roles where one role strictly dominates the other*. Our analytical results for this SoD Rule are in the third subsection of the following section, titled 'SoD Rule 3: CoI Due to Role Dominance'.

CoI Due to Common Roles Among Tasks

Conflicts can also arise when two tasks in an execution chain require the same role. In those situations if the same role is assigned to the same person, the opportunity to test or reconcile the work on the first task is lost; moreover, the person can perpetrate employee fraud. For example, the tasks t_2 ('Review & approve PR for vendor & price') and t_9 ('Approve payment of vendor invoice') in Figure 2 and Table 1 are both authorization tasks. They are also on the same execution chain as can be easily verified from Figure 2. If both of these tasks are assigned to the same person, then the person authorizing a vendor and price is also authorizing payments to the vendor. That increases the risk of vendor fraud and kickbacks. Such conflicts of interest can be avoided altogether simply by requiring that the two tasks be assigned to different persons. Therefore, our fourth SoD Rule 4 (CoI due to common roles among tasks) states the following: *for any two tasks in an execution chain and any role that is needed for both the tasks, the role should be assigned to two different people*. Our analytical results for this SoD Rule are in the fourth subsection of the following section, titled 'SoD Rule 4: CoI Due to Common Roles Among Tasks'.

FORMAL ANALYSIS OF SoD RULES

In this section, we first state each SoD rule informally and then express it formally using the definitions in the second section of the paper and in Appendix A. For each SoD rule, we address two problems. The first problem is to determine whether a given assignment α is valid and satisfies the corresponding SoD rule. The second problem is to determine whether there exists an assignment α that is valid and satisfies the corresponding SoD rule. Our results show that all but one of these problems are efficiently solvable. For the problem which is computationally intractable (that is, **NP**-complete), we present an integer linear programming (ILP) formulation that can be solved in practice using public domain ILP solvers. We consider each SoD rule in a separate subsection.

The following definition of a viable assignment is used throughout this paper.

Definition 1: *An assignment α is **viable** under a given SoD rule if α is valid and satisfies the rule.*

Validity does not ensure that assignments are consistent with SoD rules; it only ensures that for each task-role pair assigned to a person p, the role set of p includes the role. Viability, on the other hand, ensures that assignments are valid and comply with the SoD rules.

SoD Rule 1: CoI Due to Task Type

The first SoD rule that we consider imposes a restriction on the type of tasks that can be assigned to any person. We begin with informal and formal statements of the rule.

SoD Rule 1: (CoI Due to Task Type) No person should be assigned a role in more than one type of task within a business process. In other words, if a person p is assigned roles in two or more tasks, then all such tasks must be of the same type.

A Formal Statement of SoD Rule 1: For each person p, let $T_p \subseteq T$ be the set of tasks in which p is assigned a role by α; that is,

$$T_p = \{t : \text{there is an } r \in R \text{ for which } \alpha(t,r) = p\}.$$

A **valid assignment α satisifies SoD Rule 1** iff for every person $p \in P$ and every pair of tasks t_i and t_j in T_p, $\lambda(t_i) = \lambda(t_j)$.

We now present our results for SoD Rule 1. First, we show that testing whether a given assignment α is viable under SoD Rule 1 can be done efficiently.

Theorem 1: *There is a polynomial time algorithm to determine whether a given assignment α is viable under SoD Rule 1.*

Proof: Our procedure for solving this problem is shown in Algorithm 3.

Algorithm 3. Testing whether a given assignment α is viable under SoD Rule 1

```
1  Check if α is valid using Algorithm 1. If α is invalid, output "Assignment α is
   invalid" and stop.
   /* Check whether each person is assigned at most one type of
   task. */
2  Let W[1 . . | P|] be an array where each entry is initialized to –1. (See text for
   the significance of the W array.)
3  for each entry (t,r) ∈ 𝔹 do
4  |  Let p = ALPHA[t, r].
5  |  if (W[p] = − 1) then
6  |  |  W[p] = λ(t).
7  |  else if (W[p] ≠ λ(t)) then
8  |  |  output "Assignment α is not viable under SoD Rule 1" and stop.
9  |  end
10 end
11 output "Assignment α is viable under SoD Rule 1".
```

The algorithm first checks whether α is valid using Algorithm 1. If α is valid, it checks whether all the tasks assigned to each person are of the same type. To do this efficiently, the algorithm uses an array **W** with $|P|$ elements such that each entry **W**[p] stores the type of task assigned to person p. We use the convention that if no task has been assigned to p, then **W**[p] = −1. Each entry of the array **W** is initialized to –1. It is assumed that the assignment α is given by the matrix ALPHA. The algorithm goes through each task-role pair $(t,r) \in \mathbb{B}$. Suppose ALPHA[t, r] = p and the type of task t is $\lambda(t)$. If **W**[p] = –1 (that is, t is the first task assigned to p), the algorithm sets **W**[p] = $\lambda(t)$. Otherwise, it checks whether **W**[p] = $\lambda(t)$. If it is the case, the algorithm considers the next entry of $\mathbb{B}$; if not, it reports that SoD Rule 1 is not satisfied and stops. Thus, the correctness of the algorithm is evident.

To estimate the running time of Algorithm 3, note that from Lemma 1, checking whether α is valid can be done in $O(|\mathbb{B}|)$ time after reading in the ALPHA matrix. The time to read the ALPHA matrix is $O(|T| \times |R|)$ and the time to initialize the **W** array is $O(P)$. To check whether α satisfies SoD Rule 1, we note that each iteration of the **for** loop in Step 3 uses $O(1)$ time. Thus, the time used by the loop is $O(\mathbb{B})$. Hence, the running time over all the steps is $O(|\mathbb{B}| + |P| + |T| \times |R|)$. Since $|\mathbb{B}| \leq |T| \times |R|$, the running time can be simplified to $O(|P| + |T| \times |R|)$. Since P and ALPHA are all inputs to the problem, the running time of the algorithm is linear in the input size. □

EXAMPLE 4

Consider the matrices TR, RP, and ALPHA in Tables 2, 4, and 6 respectively. First, we set up an array W[1 . . $|P|$] and initialize all components to –1. The algorithm tests each (t, r) pair in $\mathbb{B}$ for viability. The results are shown in Table 10. For example, the pair (t_1, r_1), $\lambda(t_1)$ is **E**, and the task-role pair assignment for t_1 from Table 5 is p_4. Since all components of W are set initially set to –1, we have W [1] = −1, and so this value is reset to $\lambda(t_1) = \mathrm{E}$. Since in all 11 iterations $W[p] = -1$, they are replaced by the task type. The algorithm finds that all the jobs assigned to each person are of the same type and concludes that the given assignment is viable under SoD Rule 1.

Now, consider a modified assignment in which tasks t_6 and t_7 are both assigned to person p_6. The matrix RP shows that both tasks are in the role set of p_6. So, the assignment is valid. Table 11 shows the working of Algorithm 3. The first six iterations work exactly as in Table 10. During the seventh iteration, the algorithm notices that person p_6 is assigned both tasks t_6 and t_7 which are of different types. Thus, the algorithm concludes that the given assignment is *not* viable under SoD Rule 1.

When all tasks are of the same type, any valid assignment α is trivially viable under SoD Rule 1. Thus, for this special case, the problem of determining whether there is a viable assignment under SoD Rule 1 reduces to the problem of merely checking whether there is a valid assignment; by Part (ii) of Lemma 1, this special case can be solved efficiently. However, with just two types of tasks, we

TABLE 10

WORKING OF ALGORITHM 3 WHERE THE GIVEN ASSIGNMENT IS VIABLE

Iter. No.	(t, r) pair	$\lambda(t)$	$\alpha(t, r) = p$	Viability $W[p]$
1	(t_1, r_1)	E	p_4	$W[p_4] = -1.$ $\therefore Post\ W[p_4] = \lambda(t_1) = \mathtt{E}$
2	(t_2, r_2)	A	p_3	$W[p_3] = -1.$ $\therefore Post\ W[p_3] = \lambda(t_2) = \mathtt{A}$
3	(t_3, r_3)	A	p_2	$W[p_2] = -1.$ $\therefore Post\ W[p_2] = \lambda(t_3) = \mathtt{A}$
4	(t_4, r_4)	E	p_1	$W[p_1] = -1.$ $\therefore Post\ W[p_3] = \lambda(t_4) = \mathtt{E}$
5	(t_5, r_5)	E	p_{11}	$W[p_{11}] = -1.$ $\therefore Post\ W[p_{11}] = \lambda(t_5) = \mathtt{E}$
6	(t_6, r_6)	C	p_6	$W[p_6] = -1.$ $\therefore Post\ W[p_6] = \lambda(t_6) = \mathtt{C}$
7	(t_7, r_7)	R	p_7	$W[p_7] = -1.$ $\therefore Post\ W[p_7] = \lambda(t_7) = \mathtt{R}$
8	(t_8, r_8)	R	p_9	$W[p_9] = -1.$ $\therefore Post\ W[p_9] = \lambda(t_8) = \mathtt{R}$
9	(t_9, r_9)	A	p_{10}	$W[p_{10}] = -1.$ $\therefore Post\ W[p_{10}] = \lambda(t_9) = \mathtt{A}$
10	(t_{10}, r_{10})	A	p_{12}	$W[p_{12}] = -1.$ $\therefore Post\ W[p_{12}] = \lambda(t_{10}) = \mathtt{A}$
11	(t_{11}, r_{11})	E	p_5	$W[p_5] = -1.$ $\therefore Post\ W[p_5] = \lambda(t_{11}) = \mathtt{E}$

Conclusion: Assignment is viable under SoD Rule 1.

TABLE 11

WORKING OF ALGORITHM 3 WHERE THE ASSIGNMENT IS NOT VIABLE

Iter. No.	(t, r) pair	$\lambda(t)$	$\alpha(t, r) = p$	Viability $W[p]$
1	(t_1, r_1)	E	p_4	$W[p_4] = -1.$ $\therefore Post\ W[p_4] = \lambda(t_1) = \mathtt{E}$
2	(t_2, r_2)	A	p_3	$W[p_3] = -1.$ $\therefore Post\ W[p_3] = \lambda(t_2) = \mathtt{A}$
3	(t_3, r_3)	A	p_2	$W[p_2] = -1.$ $\therefore Post\ W[p_2] = \lambda(t_3) = \mathtt{A}$
4	(t_4, r_4)	E	p_1	$W[p_1] = -1.$ $\therefore Post\ W[p_1] = \lambda(t_4) = \mathtt{E}$
5	(t_5, r_5)	E	p_{11}	$W[p_{11}] = -1.$ $\therefore Post\ W[p_{11}] = \lambda(t_5) = \mathtt{E}$
6	(t_6, r_6)	C	p_6	$W[p_6] = -1.$ $\therefore Post\ W[p_6] = \lambda(t_6) = \mathtt{C}$
7	(t_7, r_7)	R	p_6	$W[p_6] = \mathtt{C}.$ *$\therefore$Assignment α is not viable under SoD Rule 1 since* $\lambda(t_7) = \mathtt{R}$.

Conclusion: Assignment is *not* viable under SoD Rule 1.

demonstrate in Theorem 2 that determining whether there is a viable assignment under SoD Rule 1 is computationally intractable. (As mentioned earlier, auditors have used four task types in analyzing SoD.) To cope with this intractability, we provide an integer linear programming formulation for the problem of finding an assignment that is viable under SoD Rule 1.

Theorem 2: *Given a business process, the problem of determining whether there is a viable assignment under SoD Rule 1 is* ***NP****-complete even when there are only two types of tasks.*

Proof: See Appendix B.1.

An integer linear programming formulation Theorem 2 points out that in general, determining whether a viable assignment exists under SoD Rule 1 is computationally intractable. Hence, there is no efficient algorithm for the problem unless the

complexity classes **P** and **NP** are equal (Garey and Johnson, 1979). We now present a {0,1} integer linear programming (ILP) formulation for the problem so that it can be solved in practice using public domain ILP solvers such as Gurobi (2018). Since our goal is to check whether there exists a viable assignment under SoD Rule 1, our formulation does not involve any optimization objective. When the ILP has a solution, we will explain how a viable assignment can be found efficiently from any solution to the ILP.

(a) Variables: For each person i ($1 \le i \le |P|$) and task type j ($1 \le j \le \tau$, where τ is the number of task types), we use a {0, 1}-valued variable x_{ij} which takes on the value 1 if person i is assigned a task of type j; otherwise, the value of x_{ij} is 0. (Thus, the number of variables is $\tau\,|P|$.)

(b) Constraints: The linear constraints to be satisfied by the above variables are given below.

(i) Under SoD Rule 1, each person must be assigned tasks of at most one type. This leads to the following set of constraints:

$$\sum_{j=1}^{\tau} x_{ij} \le 1 \quad \text{for each } i,\ 1 \le i \le |P| \tag{6}$$

(ii) Let $T_j \subseteq T$ denote the subset of all tasks whose type is j, $1 \le j \le \tau$. Further, let R_j denote the union of the role sets of the tasks in T_j. (Thus, R_j is the set of roles needed to complete all the tasks in T_j. We note that R_j can be computed from the TR matrix.) For each $r \in R_j$, let $P_r \subseteq P$ denote the subset of people who can play role r. (Note that P_r can be obtained from the RP matrix.) We need to ensure that each role in R_j is assigned to a person i such that i is assigned to one or more tasks of type j and r appears in the role set of i. This leads to the following set of constraints:

$$\sum_{i \in P_r} x_{ij} \ge 1 \quad \text{for each role } r \in R_j,\ \ 1 \le j \le \tau \tag{7}$$

This completes the specification of the ILP for Assign-R1. It can be verified that there is a viable assignment under SoD Rule 1 if there is a solution to the above set of constraints.

When there is a solution to the above ILP, a viable assignment α can be constructed from the values of the variables x_{ij} as follows. We consider each task type j ($1 \le j \le \tau$) separately. As mentioned above, let T_j denote the subset of all the tasks of type j and let P_j denote the subset of people who have been assigned tasks of type j. Since all the tasks in T_j are of the same type, assigning each role r needed for the tasks in T_j to a person in P_j whose role set includes r gives a viable assignment under SoD Rule 1. The constraints specified by equation (7) ensure that for each j, each role needed for T_j can be assigned to someone in P_j. This method of constructing a viable assignment α from the solution to the ILP is summarized in Algorithm 4.

Algorithm 4. Finding a viable assignment under SoD Rule 1 from a solution to the ILP

```
1 for j = 1 to τ do
2 |  Let P_j = {i : x_ij = 1}. (See text for the significance of P_j and T_j.)
3 |  Using the TR matrix, find R_j, the set of all roles needed for the tasks in T_j.
4 |  for each role r ∈ R_j do
5 |  |  Find a person p ∈ P_j whose role set includes r.
6 |  |  For each t ∈ T_j such that task t has role r, let α(t, r) = p.
7 |  end
8 end
9 output Assignment α constructed above.
```

SoD Rule 2: CoI Due to Role Conflicts

The SoD rule considered in this section imposes a restriction on the number of roles that can be assigned to any person. We begin with informal and formal statements of the rule.

SoD Rule 2: (CoI Due to Role Conflicts) A person should have at most one assigned role within a business process.

A Formal Statement of SoD Rule 2: For each person p, let $R_p \subseteq R$ be the set of roles assigned by α; that is,

$$R_p = \{r : \text{there is a } t \in T \text{ for which } \alpha(t,r) = p\}.$$

A **valid assignment α satisifies SoD Rule 2** iff for every person $p \in P$, $|R_p| \leq 1$.

We now present our results for SoD Rule 2.

Theorem 3: *The following problems can be solved in polynomial time: (i) determining whether a given assignment α is viable under SoD Rule 2 and (ii) given a business process, determining whether there is a viable assignment under SoD Rule 2.*

Proof: See Appendix B.2.

SoD Rule 3: CoI Due to Role Dominance

This SoD rule involves pairs of tasks which are of the same type and for which there is a directed path with one or more edges in the precedence dag D of the business process. The rule also involves pairs of roles which have the strict

domination relationship defined in the second section of this paper. Since D is acyclic, the reader should bear in mind that there is no directed path with one or more edges from a task t to itself in D. Further, we use the phrase 'a person is assigned a task' to mean that the person is assigned one or more roles in that task. Informal and formal statements of the SoD rule considered in this section are as follows.

SoD Rule 3: (CoI Due to Role Dominance) Any two persons who are assigned tasks of the same type within a directed path should not be assigned roles where one role strictly dominates the other.

A Formal Statement of SoD Rule 3: A **valid assignment α satisfies SoD Rule 3** iff for any pair of tasks t_i and t_j and for any pair of roles r_x and r_y that satisfy all the following four conditions:

(i) there is a directed path from t_i to t_j in G,
(ii) $\lambda(t_i) = \lambda(t_j)$ (that is, t_x and t_y are of the same type),
(iii) $r_x \in f_{\mathrm{TR}}(t_i)$ and $r_y \in f_{\mathrm{TR}}(t_j)$ (that is, roles r_x and r_y are required for tasks t_i and t_j respectively), and
(iv) one of the pairs (r_x, r_y) or (r_y, r_x) appears in $<_{\mathrm{R}}$ (that is, one of r_x and r_y strictly dominates the other in the role hierarchy), the assignment α satisfies the condition $\alpha(t_i, r_x) = \alpha(t_j, r_y)$.

To develop our results for SoD Rule 3, we first define a graph representation (called the auxiliary graph) that allows us to identify a simple graph theoretic property that captures SoD Rule 3.

Definition 2:

*(a) two tasks t_i and t_j are **linked** if (i) t_i and t_j are of the same type and (ii) there is a directed path with one or more edges in the precedence dag from the node representing t_i to the one representing t_j or vice versa;*
*(b) two roles r_x and r_y are **linked** if one of r_x and r_y strictly dominates the other (that is, either (r_x, r_y) or (r_y, r_x) appears in $<_{\mathrm{R}}$);*
*(c) given a task t and a role r, (t, r) is a **relevant task-role pair** (or RTR pair) if r appears in the role set of t (note that the binary relation $\mathbb{B}$ gives all the RTR pairs);*
*(d) two RTR pairs (t_i, r_x) and (t_j, r_y) are **linked** if tasks t_i and t_j are linked and roles r_x and r_y are linked.*

The following observation is a direct consequence of the above definition and the formal specification of SoD Rule 3.

Observation 1. *Let (t_i, r_x) and (t_j, r_y) be two linked RTR pairs in a business process. Any assignment that satisfies SoD Rule 3 must assign the same person to both the RTR pairs.* □

We now define a graph which is useful in developing the main results of this subsection.

Definition 3: *The **auxiliary graph** $H(V_H, E_H)$ of a business process is an undirected graph defined as follows. The node set V_H is in one-to-one correspondence with the set of RTR pairs of the business process. An edge $\{u, v\}$ is in E_H if and only if the RTR pairs corresponding to u and v are linked.*

The following lemma uses the auxiliary graph to generalize Observation 1.

Lemma 2: *Let $H(V_H, E_H)$ be the auxiliary graph of a business process. Suppose there is a path $\pi = \langle u_1, u_2, \ldots, u_r \rangle$ in H with one or more edges. Then in any assignment that satisfies SoD Rule 3, all the RTR pairs corresponding to the nodes in π must be assigned to the same person.*

Proof: We use simple induction on the number of edges in the path. The basis is for a path with one edge $\{u_1, u_2\}$. In this case, by Observation 1, the RTR pairs corresponding to u_1 and u_2 must be assigned to the same person. So, assume that for some $k \geq 1$, the result holds for all paths with at most k edges in H. Now, consider a path $\pi' = \langle u_1, u_2, \ldots, u_k, u_{k+1} \rangle$ with $k + 1$ edges. By the inductive hypothesis, all the RTR pairs corresponding to u_1 through u_k must be assigned to the same person to satisfy SoD Rule 3. Further, by Observation 1, the RTR pairs corresponding to u_k and u_{k+1} must also be assigned to the same person. Hence, the RTR pairs corresponding to all the nodes in π' must be assigned to the same person. This completes the inductive proof. □

A simple consequence of Lemma 2 is the following: any assignment that satisfies SoD Rule 3 must assign all the RTR pairs corresponding to the nodes of any *connected component* of the auxiliary graph to the same person. It can also be seen that any assignment that assigns the same person for all the RTR pairs corresponding to all the nodes in each connected component of the auxiliary graph satisfies SoD Rule 3. These two observations lead to the following result for testing whether or not a given assignment satisfies SoD Rule 3.

Corollary 1: *An assignment α satisfies SoD Rule 3 iff for each connected component H' of the auxiliary graph H, α assigns all the RTR pairs corresponding to the nodes in H' to the same person.* □

We now prove the main results of this subsection.

Theorem 4: *The following problems can be solved in polynomial time: (i) determining whether a given assignment α is viable under SoD Rule 3 and (ii) given a business process, determining whether there is an assignment that is viable under SoD Rule 3.*

Algorithm 5. Testing whether a given assignment α is viable under SoD Rule 3

1 Check if α is valid using Algorithm 1. If α is invalid, **output** "Assignment α is invalid" and **stop**.
2 Using the relevant RTR pairs (given by the relation $\mathbb{B}$), construct the auxiliary graph H.
3 Find the connected components of H.
4 for *each connected component* H' *of* H **do**
5 Check whether α assigns all the RTR pairs corresponding to the nodes of H' to the same person p. If not, **output** "Assignment α does not satisfy SoD Rule 3" and **stop**.
6 end
7 output "Assignment α is viable under SoD Rule 3".

Proof of Part (i): Our procedure to test whether a given assignment α is viable under SoD Rule 3 is shown in Algorithm 5. The algorithm first checks whether α is valid. It then checks whether the given assignment satisfies SoD Rule 3 using Corollary 1. So, the correctness of the algorithm is obvious.

To estimate its running time, recall that Step 1 can be done in $O(|\mathbb{B}|)$ time (Part (i) of Lemma 1). The auxiliary graph H has $|V_H| = |\mathbb{B}|$ nodes, and thus at most $O\left(|\mathbb{B}|^2\right)$ edges. As stated in Appendix A, we assume that checking whether there is a directed path between a given pair of tasks in the precedence dag and checking whether there is a strict domination relationship between a given pair of roles can be done in $O(1)$ time. Hence, the auxiliary graph construction (Step 2) can be carried out in $O\left(|\mathbb{B}|^2\right)$ time. Since H has $|\mathbb{B}|$ nodes and $O\left(|\mathbb{B}|^2\right)$ edges, finding the connected components of H (Step 3) can be done in $O\left(|\mathbb{B}|^2\right)$ time. For each connected component H' of H with $N_{H'}$ nodes, we can check (using the matrix representation ALPHA of α) whether α assigns all the RTR pairs corresponding to the nodes of H' to the same person in time $O(N_{H'})$. Therefore, over all the connected components, the time used by the algorithm in Step 4 is $O(|V_H|) = O(|\mathbb{B}|)$. So, the running time of the algorithm is $O\left(|\mathbb{B}|^2\right)$.

We now present examples to illustrate Algorithm 5.

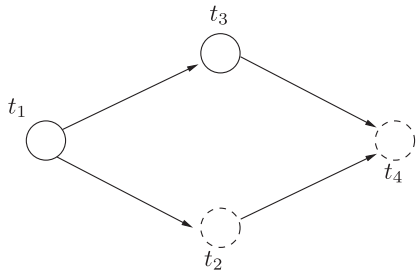
EXAMPLE 5

The set T of tasks for the business process considered here has four tasks; that is, $T = \{t_1, t_2, t_3, t_4\}$. Of these, t_1 and t_3 are of Type 1 while t_2 and t_4 are of Type 2. The set of roles R and people are given by $R = \{r_1, r_2, r_3, r_4\}$ and $P = \{p_1, p_2, p_3, p_4\}$. For simplicity, we will assume that the role-person (RP) matrix has all its entries to be 1; that is, each person in P can be assigned any of the roles in R. The dependency graph and the task-role (TR) matrix for this business

FIGURE 4

DEPENDENCY GRAPH AND TR MATRIX FOR EXAMPLE 5

Dependency Graph



TR Matrix

	r_1	r_2	r_3	r_4
t_1	1	1	0	0
t_2	0	1	1	0
t_3	0	0	1	1
t_4	1	0	0	1

Note: In the dependency graph, two forms of circles to show that tasks t_1 and t_3 are of Type 1 while tasks t_2 and t_4 are of Type 2.

process are shown in Figure 4. Let the partial order $<_R$ on the roles be given by

$$<_R = \{(r_1,r_3),(r_1,r_4),(r_2,r_4),(r_3,r_4)\}$$

From the TR matrix, the set $\mathbb{B}$ of relevant task-role pairs (that is, RTR pairs) is given by

$$\mathbb{B} = \{(t_1,r_1),(t_1,r_2),(t_2,r_2),(t_2,r_3),(t_3,r_3),(t_3,r_4),(t_4,r_1),(t_4,r_4)\}$$

For this example, the set of linked pairs of tasks is $\{(t_1, t_3), (t_2, t_4)\}$ and the set of linked roles is the same as $<_R$. Using this information, the auxiliary graph H constructed in Step 2 of Algorithm 5 is shown in Figure 5.

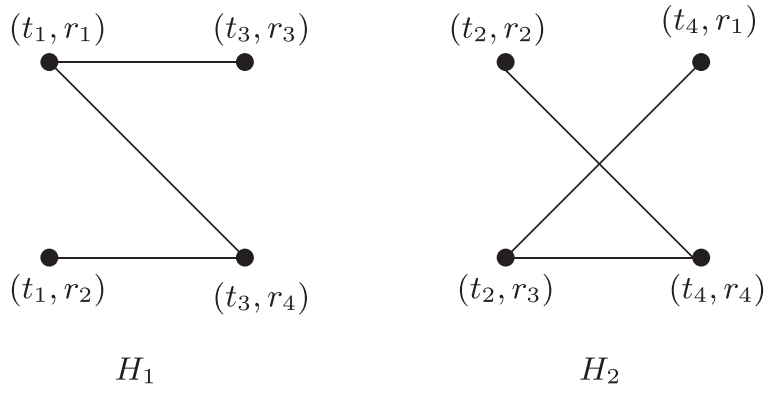
Let us first consider the following assignment α for this business process.

$$\alpha(t_1,r_1) = p_1\ \alpha(t_1,r_2) = p_2\ \alpha(t_2,r_2) = p_3\ \alpha(t_2,r_3) = p_4$$
$$\alpha(t_3,r_3) = p_1\ \alpha(t_3,r_4) = p_2\ \alpha(t_4,r_1) = p_3\ \alpha(t_4,r_4) = p_4$$

The above assignment α is valid since each person can play any of the four roles. For an assignment to be viable under SoD Rule 2, it must be valid and for each connected component of the auxiliary graph, all the RTR pairs in that component must be assigned to the same person. In the above assignment, even though the two RTR pairs (t_1, r_1) and (t_1, r_2) are in the same connected component (namely, H_1) of the auxiliary graph H, the given assignment α assigns those two RTR pairs to two different people, namely p_1

FIGURE 5

THE AUXILIARY GRAPH H FOR EXAMPLE 5. THE GRAPH CONSISTS OF TWO CONNECTED COMPONENTS, DENOTED BY H_1 AND H_2



and p_2. Thus, Algorithm 5 outputs the message 'Assignment α does not satisfy SoD Rule 3'.

EXAMPLE 6

For the business process in Example 5, let us consider the following assignment α_1 (instead of α).

$$\alpha_1(t_1,r_1)=p_2\ \alpha_1(t_1,r_2)=p_2\ \alpha_1(t_2,r_2)=p_3\ \alpha_1(t_2,r_3)=p_3$$
$$\alpha_1(t_3,r_3)=p_2\ \alpha_1(t_3,r_4)=p_2\ \alpha_1(t_4,r_1)=p_3\ \alpha_1(t_4,r_4)=p_3$$

As before, α_1 is also a valid assignment. Further, all the RTR pairs in the connected component H_1 are assigned to p_2 while all the RTR pairs in the connected component H_2 are assigned to p_3. Thus, Algorithm 5 outputs the message 'Assignment α is viable under SoD Rule 3'.

Proof of Part (ii): Our method to determine whether there is a viable assignment under SoD Rule 3 is shown in Algorithm 6. When the answer is 'yes', the algorithm also outputs one such assignment.

Since the algorithm merely implements the condition mentioned in Corollary 1, its correctness is obvious. We now estimate its running time. As mentioned in the proof of Part (i), Steps 1 and 2 can be done in $O\left(|\mathbb{B}|^2\right)$ time. Let γ denote the number of connected components of H. For each connected component H' with $N_{H'}$ nodes, we can find the set R' of roles used in the RTR

Algorithm 6. Finding a viable assignment α under SoD Rule 3

1 Using the relevant RTR pairs (given by the relation $\mathbb{B}$), construct the auxiliary graph H.

2 Find the connected components of H.

3 for *each connected component* H' *of* H **do**

4 Let R' be the set of roles needed to complete the tasks corresponding to the RTR pairs represented by the nodes in H'.

5 Check (using the RP matrix) whether there is a person p such that the role set of p includes every role in R'.

6 If so, for each RTR pair (t, r) that corresponds to a node in H', let $\alpha(r, t) = p$; otherwise, **output** "There is no viable assignment" and **stop**.

7 end

8 output Assignment α constructed above.

pairs of H' in $O(|R| + N_{H'})$ time using a vector of size $|R|$ in a manner similar to that used in the proof of Part (i) of Theorem . Since $N_{H'} \leq |\mathbb{B}|$, the time used for this computation is $O(|R| + |\mathbb{B}|)$. Using the RP matrix, we can check whether there is a person p whose role set is a superset of R' in $O(|R| \times |P|)$ time. Thus, the time used by the algorithm for each iteration of the loop in Step 3 is $O(|R| \times |P|)$. Hence, the total time for Step 3 is $O(\gamma(|R| + |\mathbb{B}| + |R| \times |P|)) = O(\gamma(|\mathbb{B}| + |R| \times |P|))$. Step 8 can be done in $O(|\mathbb{B}|)$ time. So, the running time of the algorithm is $O\left(|\mathbb{B}|^2 + \gamma(|\mathbb{B}| + |R| \times |P|)\right)$. Since $\gamma \leq |\mathbb{B}|$, the running time of the algorithm can be simplified to $O\left(|\mathbb{B}|^2 + |\mathbb{B}| \times |R| \times |P|\right)$, which is a polynomial function of the input size. □

SoD Rule 3 used a particular binary relationship between RTR pairs; for two RTR pairs (t_i, r_x) and (t_j, r_y), this relationship requires the corresponding pair of tasks and roles to be linked. The polynomial time algorithms presented in the proofs of Part (i) and Part (ii) of Theorems 4 can be generalized to SoD Rules based on any binary relation on the set of RTR pairs, as long as one can efficiently determine whether two given RTR pairs are related.

SoD Rule 4: CoI Due to Common Roles Among Tasks

Recall from the earlier section on SoD Rule 3, the notion of linked tasks: two tasks t_i and t_j are linked if they are of the same type and there is a directed path from t_i to t_j or *vice versa* in the precedence dag. SoD Rule 4 also

concerns linked tasks. Informal and formal statements of this rule are as follows.

SoD Rule 4: (CoI Due to Common Roles Among Tasks) For any two linked tasks and any role that is needed for both the tasks, the role should be assigned to two different people.

A Formal Statement of SoD Rule 4: A **valid assignment α satisfies SoD Rule 4** iff the following condition holds: for any pair of tasks t_i and t_j and any role r such that (i) there is a directed path from t_i to t_j in the precedence dag D, (ii) $\lambda(t_i) = \lambda(t_j)$ (that is, t_i and t_j are of the same type) and (iii) $r \in f_{\mathrm{TR}}(t_i) \cap f_{\mathrm{TR}}(t_j)$ (that is, role r is required in both t_i and t_j), the assignment α satisfies the condition $\alpha(t_i, r) \neq \alpha(t_j, r)$.

We first show that there is an efficient algorithm to check whether a given assignment α is viable under SoD Rule 4.

Theorem 5: *There is a polynomial time algorithm to determine whether a given assignment α is viable under SoD Rule 4.*

Proof: Our method of checking whether a given assignment α is viable under SoD Rule 4 is shown in Algorithm 7.

The correctness of the algorithm is obvious since it directly implements the test mentioned in the formal statement of SoD Rule 4. To estimate its running time, recall that testing whether a given assignment is valid (Step 1) can be done in $O(|\mathbb{B}|)$ time. The number of iterations of the loop in Step 2 is at most $|T|^2$. In each iteration, determining whether two tasks are linked can be done in $O(1)$ time

Algorithm 7. Testing whether a given assignment α is viable under SoD Rule 4

```
1  Check if α is valid using Algorithm 1. If α is invalid, output "Assignment α is
     invalid" and stop.
2  for each pair t_i and t_j of linked tasks do
3  |  Find the set R_ij of roles needed by both t_i and t_j.
4  |  for each role r ∈ R_ij do
5  |  |  if (α(t_i, r) = α(t_j, r)) then
6  |  |  |  output "Assignment α does not satisfy SoD Rule 4" and stop.
7  |  |  end
8  |  end
9  end
10 output "Assignment α is viable under SoD Rule 4".
```

using the preprocessing steps mentioned in Appendix A2. For a pair of tasks t_i and t_j, the set R_{ij} of roles that are needed for both t_i and t_j can be found in $O(|R|)$ time using the TR matrix. For each role $r \in R_{ij}$, the test in the body of the loop in Step 4 can be done in $O(1)$ time using the matrix representation of the given assignment α. Thus, the total time used by the loop in Step 4 is $O(|R|)$. Hence, the time for each iteration of the loop in Step 2 is $O(|R|)$. So, the total time for Step 2 is $O(|T|^2 \times |R|)$. Therefore, the running time of the algorithm is $O\left(|\mathbb{B}| + |T|^2 \times |R|\right)$. Since $|\mathbb{B}| \leq |T| \times |R|$, the running time can be simplified to $O(|T|^2 \times |R|)$, which is a polynomial function of the input size. □

We now present examples to illustrate Algorithm 7.

EXAMPLE 7

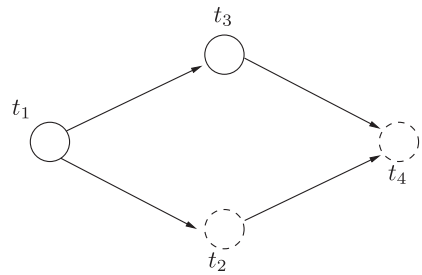
The set T of tasks for the business process considered here has four tasks; that is, $T = \{t_1, t_2, t_3, t_4\}$. Of these, t_1 and t_3 are of type 1 while t_2 and t_4 are of type 2. The set of roles R and people are given by $R = \{r_1, r_2, r_3, r_4\}$ and $P = \{p_1, p_2, p_3, p_4\}$. For simplicity, we will assume that the role-person (RP) matrix has all its entries as 1; that is, every person in P can be assigned any of the roles in R. The dependency graph (which is the same as the one used to illustrate Algorithm 5) and the task-role (TR) matrix for this business process are shown in Figure 6. From the TR matrix, the set $\mathbb{B}$ of task-role pairs is given by $\mathbb{B} = \{(t_1,r_1),(t_1,r_2),(t_2,r_2),(t_2,r_3),(t_2,r_4),(t_3,r_1),(t_3,r_2),(t_3,r_4),(t_4,r_3),(t_4,r_4)\}$.

For this example, the set of linked pairs of tasks is $\{(t_1, t_3), (t_2, t_4)\}$. Consider the following assignment α.

FIGURE 6

DEPENDENCY GRAPH AND TR MATRIX FOR EXAMPLE 7

Dependency Graph



TR Matrix

	r_1	r_2	r_3	r_4
t_1	1	1	0	0
t_2	0	1	1	1
t_3	1	1	0	1
t_4	0	0	1	1

Note: In the dependency graph, two forms of circles to show that tasks t_1 and t_3 are of Type 1 while tasks t_2 and t_4 are of Type 2.

$$\alpha(t_1,r_1)=p_1\ \alpha(t_1,r_2)=p_2\ \alpha(t_2,r_2)=p_3\ \alpha(t_2,r_3)=p_4\ \alpha(t_2,r_4)=p_4$$
$$\alpha(t_3,r_1)=p_1\ \alpha(t_3,r_2)=p_2\ \alpha(t_3,r_4)=p_3\ \alpha(t_4,r_3)=p_4\ \alpha(t_4,r_4)=p_4$$

The above assignment α is valid since each person can play any of the four roles. For an assignment to be viable under SoD Rule 4, it must be valid and for each pair (t_i, t_j) of linked tasks and each role r that is needed in both t_i and t_j, the people assigned the role in t_i and t_j must be different. In the above example, (t_1, t_3) is a linked pair of tasks and role r_1 is needed by both t_1 and t_3. However, the assignment α chooses the same person (namely, p_1) for the role r_1 in both t_1 and t_3. Thus, Algorithm 7 would output the message 'Assignment α does not satisfy SoD Rule 4'.

EXAMPLE 8

For the business process in Example 7, consider the following assignment α_1.

$$\alpha_1(t_1,r_1)=p_1\ \alpha_1(t_1,r_2)=p_2\ \alpha_1(t_2,r_2)=p_3\ \alpha_1(t_2,r_3)=p_4\ \alpha_1(t_2,r_4)=p_1$$
$$\alpha_1(t_3,r_1)=p_2\ \alpha_1(t_3,r_2)=p_3\ \alpha_1(t_3,r_4)=p_3\ \alpha_1(t_4,r_3)=p_2\ \alpha_1(t_4,r_4)=p_4$$

The above assignment α_1 is valid since each person can play any of the four roles. Algorithm 7 first considers the linked pair of tasks (t_1, t_3) and notices that roles r_1 and r_2 appear in both of these tasks. However, α_1 assigns (t_1, r_1) to p_1 and (t_3, r_1) to p_2. Further, α_1 assigns (t_1, r_2) to p_2 and (t_3, r_2) to p_3. Thus, the condition for SoD Rule 3 is satisfied for the linked pair of tasks (t_1, t_3). In a similar manner, it can be verified that the condition for SoD Rule 3 is satisfied for the other linked pair of tasks, namely (t_2, t_4). Thus, Algorithm 7 would output the message 'Assignment α_1 is viable under SoD Rule 4'.

In the remainder of this subsection, we show (see Theorem 6) that the problem of determining whether a given business process has a viable assignment under SoD Rule 4 can be solved efficiently. As a first step towards that result, we observe that the rule imposes constraints only on tasks of the same type; thus, we can consider each task type separately. Moreover, among the tasks of the same type, the rule applies to each role that is needed by one or more tasks of the chosen type. Thus, the problem can be further decomposed by considering each role separately within each set of tasks of the same type. Hence, we have a separate subproblem for each combination of task type and role. We state this formally below.

Observation 2. *Given a business process, there is a viable assignment under SoD Rule 4 for the business process iff there is an assignment that is viable under the rule for each subproblem specified by a task type i and role j, where role j is needed by at least one task of type i.* □

In view of Observation 2, we focus on determining whether there is a viable assignment under Rule 4 for each subproblem specified by a task type and a role. The following definitions help us to develop an efficient algorithm for a given subproblem.

Definition 4:

*(a) let $D(V, A)$ be the precedence dag of the given business process, for $1 \le i \le \tau$, the **type** i **precedence dag**, denoted by $D_i(V_i, A_i)$, is defined as follows and the node set $V_i \subseteq V$ consists of all the tasks of type i. For two nodes u and v in V_i, the directed edge (u, v) is in A_i iff there is a directed path with at least one edge from u to v in the original precedence dag D;*

*(b) let R_i denote the set of all roles needed for one or more tasks of type i. for each role $j \in R_i$, define the **type-role** (i, j) **precedence dag**, denoted by $D_i^j\left(V_i^j, A_i^j\right)$, as follows and the node set $V_i^j \subseteq V_i$ consists of all the tasks in V_i which require role j. For two nodes u and v in V_i^j, the directed edge (u, v) is in A_i^j iff there is a directed path with at least one edge from u to v in the type i precedence dag D_i.*

A direct consequence of the definition of the type-role precedence graph is the following.

Observation 3. *Consider the type-role precedence dag D_i^j. Let u and v be two distinct nodes in D_i^j. If there is a path from u to v in D_i^j, then the edge (u, v) is also in D_i^j.* □

The following lemma shows how the definition of the type-role precedence dag is useful for SoD Rule 4.

Lemma 3: *Consider the type-role precedence dag D_i^j for task type i and role j. Let u and v two distinct nodes in D_i^j such that the edge (u, v) in D_i^j. Any assignment α that is viable under SoD Rule 4 must satisfy the condition $\alpha(u, j) \neq \alpha(v, j)$.*

Proof: Since the edge (u, v) is in D_i^j, by the definition of D_i^j, there is a path from u to v in the original dag D. Further, the tasks corresponding to u and v are of the same type (namely, type i) and they both require role j. Therefore, by the definition of SoD Rule 4, it follows that for any assignment that satisfies the rule, $\alpha(u, j) \neq \alpha(v, j)$. □

We introduce an additional definition that is useful in constructing a viable assignment to a given subproblem. Given a dag $Q(V_Q, A_Q)$, the **level number** of any node $u \in V_Q$, denoted by $L(u)$, is the number of nodes in a *longest* directed path ending at u. To compute the level numbers of nodes in a dag efficiently, we use the following recursive definition.

Definition 5: *Given a dag $Q(V_Q, A_Q)$, the **level number** $L(u)$ of a node u can be computed as follows: (i) if u has no incoming edges in D_i^j, then $L(u) = 1$, and (ii) if u has one or more incoming edges in Q, then $L(u) = 1 + \max\{L(w) : (w, v) \in A_Q\}$.*

It can be verified that the two definitions of the level number given above are equivalent. The following is an observation regarding level numbers.

Observation 4. *Suppose u and v are two distinct nodes in a dag Q such that $L(u) = L(v)$. Then there is no directed path in D_i^j between u and v.* □

Since the computation of level numbers plays an important role in determining whether a given business process has an viable assignment under SoD Rule 4, we now present an efficient algorithm for that computation. This algorithm uses the following fact about dags (Cormen *et al.*, 2009).

Fact 1. *Each nonempty dag $Q(V_Q, A_Q)$ has at least one node with no incoming edges.* □

Lemma 4: *Given a dag $Q(V_Q, A_Q)$ represented by its $|V_Q| \times |V_Q|$ adjacency matrix, the level numbers of all the nodes in V_Q can be computed in $O(|V_Q|^2)$ time.*

Proof: A procedure for computing the level numbers of all the nodes in a dag $Q(V_Q, A_Q)$ is shown in Algorithm 8. In each iteration, the algorithm looks for the set of all nodes with no incoming edges. It succeeds in finding such a set because of Fact 1 and the observation that a dag continues to remain acyclic when some of the nodes and edges are removed.

Algorithm 8. Computing the level numbers of nodes in a dag $Q(V_Q, A_Q)$

1 Set $k = 1$. (Variable k is used for assigning levels to nodes in the dag $Q(V_Q, A_Q)$.)

2 while *there are nodes in V_Q* **do**

3 Find the subset Z_k consisting of all the nodes with no incoming edges. For each node $u \in Z_k$, let $L(u) = k$.

4 Remove all the nodes in Z_k from V_Q. (This step also removes all the edges incident on the nodes in Z_k.)

5 $k = k + 1$.

6 end

Thus, the set of nodes will become empty at some stage and the algorithm will terminate. It can be seen through a simple inductive argument (on the number of nodes in a longest path ending at a node) that for each node u, the value $L(u)$ computed by the algorithm is the number of nodes in a longest path ending at u.

We now discuss how Algorithm 8 can be implemented efficiently. Let $\mathbb{A}$ denote the $|V_Q| \times |V_Q|$ adjacency matrix of Q. In addition to $\mathbb{A}$, we use an array $\mathbf{C}[1\,..\,|V_Q|]$ of counters, where each entry $\mathbf{C}[u]$ stores the number of incoming edges to node u. Initially, all the entries of the $\mathbf{C}$ array can be computed in

$O(|V_Q|^2)$ time using $\mathbb{A}$; this is because the value $\mathbf{C}[u]$ is the number of 1's in the column corresponding to u in $\mathbb{A}$. During any iteration, the set Z_k of all the nodes with no incoming edges can be found in $O(|V_Q|)$ time using the $\mathbf{C}$ array; assigning the level number k to all the nodes in Z_k can also be in $O(|V_Q|)$ time. Removing the nodes in Z_k and deleting each edge incident on those nodes can be done by changing the appropriate entry of the matrix $\mathbb{A}$ from 1 to 0. When the incoming edge to a node u is deleted, the counter $\mathbf{C}[u]$ is decremented by 1. Thus, deleting each edge can be implemented to run in $O(1)$ time. Since the number of edges in Q is at most $|V_Q|^2$, the total time spent by the algorithm to assign level numbers to all the nodes is $O(|V_Q|^2)$, which is linear in the input size. □

The following lemma uses the notion of level numbers to develop a necessary and sufficient condition for the existence of a viable assignment under SoD Rule 4 for each subproblem.

Lemma 5: *Consider the dag $D_i^j(V_i^j, A_i^j)$ and suppose for each node $u \in V_i^j$, the level number $L(u)$ has been computed. Let $\ell = \max\left\{L(u) : u \in D_i^j\right\}$. There is a viable assignment under SoD Rule 4 for the subproblem defined by a task type i and role j iff the number of people who can play role j is at least ℓ.*

Proof:
(i) If part: Assume that there are at least ℓ people who can play role j. Consider a subset $P_i^j = \{p_1, p_2, \ldots, p_\ell\}$ of such people with ℓ members. Using P_i^j, we can construct an assignment α_i^j that is viable under SoD Rule 4 for the subproblem defined by tasks of type i and role j as follows. For each node $u \in V_i^j$, we set $\alpha_i^j(u,j) = p_k$, where $k = L(u)$. Using Observation 4, it can be verified that this assignment is viable under SoD Rule 4 for the subproblem under consideration.
(ii) Only If part: Suppose there is a viable assignment α_i^j under SoD Rule 4 for the subproblem defined by task type i and role j. Since ℓ is the maximum level number, there is a node u such that a longest path in D_i^j ending in u has ℓ nodes. Let $\pi = \langle u_1, u_2, \ldots, u_{\ell-1}, u_\ell = u\rangle$ be such a path. By Observation 3, for each pair of distinct nodes u_x and u_y in π, with $x<y$, D_i^j contains the edge (u_x, u_y). Since α_i^j satisfies SoD Rule 4, by Lemma 3, α_i^j cannot assign the same person for role j for any pair of tasks in the path π. Thus, α_i^j must use at least ℓ people who can play role j. □

The following lemma shows that the construction of an assignment for a given subproblem (mentioned in the proof of the 'If part' of Lemma 5) can be carried out efficiently.

Lemma 6: *Let D_i^j denote the dag corresponding to the subproblem specified by task type i and role j. Let ℓ denote the largest level number assigned by*

Algorithm 8 to a node of D_i^j. Suppose the number of people who can play role j is at least ℓ. Then a viable assignment under SoD Rule 4 for the subproblem can be constructed efficiently.

Proof: Our procedure for creating a viable assignment for the subproblem given by dag D_i^j is shown in Algorithm 9. The procedure follows the method discussed in the proof of the 'If part' of Lemma 5.

Algorithm 9. Constructing the assignment α_i^j in the Proof of Lemma 6

1 Assign level numbers to the nodes of $D_i^j(V_i^j, A_i^j)$ using Algorithm 8. Let ℓ be the maximum level number assigned to a node.
2 Let $P_i^j = \{p_1, p_2, \ldots, p_\ell\}$ denote a set of ℓ people who can play role *j*.
3 for $u \in V_i^j$ **do**
4 Let $k = L(u)$; set $\alpha_i^j(u,j) = p_k$.
5 end

As before, we assume that the directed graph D_i^j is stored using its adjacency matrix AD_i^j, which is of size $|V_i^j| \times |V_i^j|$. From Lemma 4, we know that level assignments for all nodes (Step 1) can be done in $O\left(\left|V_i^j\right|^2\right)$ time. Since $|V_i^j| \leq |T|$, the time used for level assignment is $O(|T|^2)$. Step 2 can be completed in $O(|P|)$ time using the RP matrix and Step 3 can be done in $O\left(|V_i^j|\right) = O(|T|)$ time. So, the overall running time of Algorithm 9 is $O(|T|^2+|P|)$. □

We now present our algorithm for determining whether there is a viable assignment under SoD Rule 4.

Theorem 6: *Given a business process, there is a polynomial time algorithm to determine whether there is a viable assignment under SoD Rule 4.*

Proof: The basic idea is to decompose the problem into subproblems, where each subproblem is specified by a task type and a role. For each subproblem, we apply the algorithm given in the proof of Lemma 5. We thus obtain the procedure shown in Algorithm 10 for determining whether there is a viable assignment under SoD Rule 4. The correctness of the algorithm is a direct consequence of Observation 2 and Lemma 5.

Algorithm 10. Finding a viable assignment α under SoD Rule 4

1 for $i = 1$ **to** τ **do**
2 Let R_i be the set of all roles needed to complete the tasks of type i. (R_i can be computed from the TR matrix.)
3 **for** *each* $j \in R_i$ **do**
4 Construct the type-role precedence dag D_i^j.
5 Assign level numbers to the nodes of D_i^j using Algorithm 8. Let ℓ be the largest level number assigned.
6 **if** (*the number of people who can play role j is* $\geq \ell$) **then**
7 Construct assignment α_i^j using Algorithm 9.
8 **else**
9 **output** "There is no viable assignment" and **stop**.
10 **end**
11 **end**
12 end
13 Construct the assignment α by combining the assignments for each subproblem computed above.
14 **output** the assignment α.

TABLE 12

SUMMARY OF RESULTS

	SoD Rule	SoD EnforcementFunction	Algorithm No.	Theorem or Lemma No.
0	No SoD Rule	Testing validity of a given assignment	1	Lemma 1(i)
		Finding a valid assignment	2	Lemma 1(ii)
1	Avoid task type conflict	Testing viability for SoD Rule 1	3	Theorem 1
		Finding a viable assignment for SoD Rule 1	4	Theorem 2
2	Avoid role conflict	Testing viability for SoD Rule 2	Similar to Algorithm 3	Theorem 3(i)
		Finding a viable assignment for SoD Rule 2	11	Theorem 3(ii)
3	Avoid dominating roles along directed paths	Testing viability for SoD Rule 3	5	Theorem 4(i)
		Finding a viable assignment for SoD Rule 3	6	Theorem 4(ii)
4	Avoid role conflicts along directed paths	Testing viability for SoD Rule 4	7	Theorem 5
		Finding a viable assignment for SoD Rule 4	10	Theorem 6

The running time of Algorithm 10 can be estimated as follows. Since each subproblem is specified by a task type and role, the number of subproblems is $\tau \mid R|$. The loop in Step 3 tries to construct a partial assignment for each subproblem. As mentioned in the proof of Lemma 6, such a partial assignment can be constructed in $O(|T|^2+|P|)$ time. Therefore, the time for solving all the subproblems is $O(\tau|R|\times(|T|^2+|P|))$. Hence, Algorithm 10 runs in $O(\tau|R|\times(|T|^2+|P|))$ time, which is a polynomial function of the input size. □

Summary of Results: The above results are summarized in Table 12.

CONCLUSIONS AND FUTURE WORK

In this paper, we provide a computational framework for the study of SoD. Our framework includes a model of accounting workflows, a set of hierarchically organized roles, and a set of four SoD rules. We address the issue of enforcing SoD rules in accounting systems by developing a set of efficient algorithms. They are enforced in the context of assignment of tasks (in accounting workflows) to people. Our paper answers two fundamental questions for any business process. First, is a given assignment SoD compliant? Second, is there a compliant assignment? We present algorithms to answer these two fundamental questions for each of the four SoD rules. We also analyze the computation time used by our algorithms.

Our results show that there are efficient algorithms for answering the first question for all four SoD rules (see Theorems 1, 3, 4, and 5). The results for the second question are mixed. We present efficient algorithms for the last three SoD rules (see Theorems 3, 4, and 6). We also show (see Theorem 2) that the problem of finding a viable assignment for SoD Rule 1 (that is, avoiding task type conflict), is computationally intractable (**NP**-complete). For that case, we provide an integer linear programming (ILP) formulation which can be used in practice to check whether a viable assignment exists, and if so, to find one such assignment. These results are summarized in Table 12.

We close by pointing out possible directions for future research. First is to investigate suitable generalizations of the SoD rules considered in this paper. For example, SoD Rule 1 can be generalized to allow a person to handle k types of tasks for some $k \geq 2$. Likewise, SoD Rule 2 can be generalized to allow a person to play r roles for some $r \geq 2$. Another direction is to formulate new SoD rules that also consider resource constraints. For example, when there are many tasks and the number of persons in an organization is small, a natural constraint is to limit the number of people assigned to each task. Finally, when there is no viable assignment for a given SoD rule, the question of how the rule can be revised to obtain a viable assignment is also an interesting research direction.

REFERENCES

Aksit, M. (1996), 'Separation and Composition of Concerns in the Object-oriented Model', *ACM Computing Surveys*, Vol. 28, No. 4.

Ashbaugh-Skaife, H., D. W. Collins, W. R. Kinney Jr., and R. LaFond (2008), 'The Effect of SOX Internal Control Deficiencies and their Remediation on Accrual Quality', *The Accounting Review,* Vol. 83, No. 1, pp. 217–50.

—— (2009), 'The Effect of SOX Internal Control Deficiencies on Firm Risk and Cost of Equity', *Journal of Accounting Research,* Vol. 47, No. 1, pp. 1–43.

Bauer, A. M. (2016), 'Tax Avoidance and the Implications of Weak Internal Controls', *Contemporary Accounting Research*, Vol. 33, No. 2, pp. 449–86.

Best, E. and J. Desel (1990), 'Partial Order Behaviour and Structure of Petri Nets', *Formal Aspects of Computing*, Vol. 2, No. 1, pp. 123–38.

Botha, R. A. and J. H. P. Eloff (2001), 'Separation of Duties for Access Control Enforcement in Workflow Environments', *IBM Systems Journal*, Vol. 40, No. 3, pp. 666–82.

Clark, D. D. and D. R. Wilson (1987), 'A Comparison of Commercial and Military Computer Security Policies', *IEEE Symposium on Security and Privacy*, pp. 184–94.

Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2009), *Introduction to Algorithms*, MIT Press and McGraw-Hill, Cambridge, MA.

Deelman, E., G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, *et al*. (2005), 'Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems', *Scientific Programming,* Vol. 13, No. 3, pp. 219–37.

Deelman, E., K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. da Silva, and M. Livny, *et al*. (2015), 'Pegasus, A Workflow Management System for Science Automation', *Future Generation Computer Systems,* Vol. 46, pp. 17–35.

Dworkin, R. (1978), *Taking Rights Seriously*, Harvard University Press, Cambridge, MA.

Elsas, P. (2008), 'X-raying Segregation of Duties: Support to Illuminate an Enterprise's Immunity to Solo-fraud', *International Journal of Accounting Information Systems*, Vol. 9, No. 2, pp. 82–93.

Gao, F., J. S. Wu, and J. Zimmerman (2009), 'Unintended Consequences of Granting Small Firms Exemptions from Securities Regulation: Evidence from the Sarbanes-Oxley Act', *Journal of Accounting Research*, Vol. 47, No. 2, pp. 459–506.

Gao, X. and Y. Jia (2016), 'Internal Control over Financial Reporting and the Safeguarding of Corporate Resources: Evidence from the Value of Cash Holdings', *Contemporary Accounting Research*, Vol. 33, No. 2, pp. 783–814.

Garey, M. R. and D. S. Johnson (1979), *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman & Co., San Francisco, CA.

Girault, C. and R. Valk (2001), *Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications*, Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Gordon, L. A. and A. L. Wilford (2012), 'An Analysis of Multiple Consecutive Years of Material Weaknesses in Internal Control', *The Accounting Review,* Vol. 87, No. 6, pp. 2027–60.

Gurobi, O. (2018), *Gurobi Optimizer Reference Manual*, Technical report, Gurobi Optimization LLC. Retrieved from http://www.gurobi.com/documentation/8.0/refman.pdf.

Hoitash, R., U. Hoitash, and K. M. Johnstone (2012), 'Internal Control Material Weaknesses and CFO Compensation', *Contemporary Accounting Research*, Vol. 29, No. 3, pp. 768–803.

Holmstrom, B. and P. Milgrom (1991), 'Multitask Principal-Agent Analyses: Incentive Contracts, Asset Ownership, and Job Design', *Journal of Law, Economics, & Organization*, Vol. 7, pp. 24–52.

Itoh, H. (1991), '*Incentives to Help in Multi-agent Situations*', *Econometrica: Journal of the Econometric Society*, pp. 611–36.

Juh´as, G., R. Lorenz, and J. Desel (2009), 'Unifying Petri Net Semantics with Token Flows', in G. F. Giuliana and K. Wolf (Eds), *Applications and Theory of Petri Nets: 30th International Conference*, Proceedings of the PETRI NETS 2009 Conference, Paris, France, 22–26 June, Springer, Berlin, pp. 2–21.

Kinney, Jr., W. R. and M. L. Shepardson (2011), 'Do Control Effectiveness Disclosures Require SOX 404 (b) internal control audits? A natural experiment with Small US Public Companies', *Journal of Accounting Research*, Vol. 49, No. 2, pp. 413–48.

Knorr, K. and H. Weidner (2001), 'Analyzing Separation of Duties in Petri Net Workflows', *Information Assurance in Computer Networks, pp.* 102–14.

Kobelsky, K. W. (2014), 'A Conceptual Model for Segregation of Duties: Integrating Theory and Practice for Manual and IT-supported Processes', *International Journal of Accounting Information Systems*, Vol. 15, No. 4, pp. 304–22.

Lægaard, J. (2006), *Organizational Theory*, bookboon.com.

Lautenbach, K. (1987), 'Linear Algebraic Techniques for Place/Transition Nets', in W. Brauer, W. Reisig, and G. Rozenberg (Eds), *Petri Nets: Central Models and their Properties – Part 1*, Lecture Notes in Computer Science, Springer-Verlag, pp. 142–67.

Liu, C. L. (1985), *Elements of Discrete Mathematics*, McGraw-Hill, Inc., New York.

Ogneva, M., K. R. Subramanyam, and K. Raghunandan (2007), 'Internal Control Weakness and Cost of Equity: Evidence from SOX Section 404 Disclosures', *The Accounting Review*, Vol. 82, No. 5, pp. 1255–1297.

Sadiq, W. and M. E. Orlowska (1999), 'Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models', *CAiSE*, Vol. 99, pp. 195–209.

Sandhu, R. S. (1988), 'Transaction Control Expressions for Separation of Duties', *IEEE Fourth Aerospace Computer Security Applications Conference*, pp. 282–86.

—— (1990), 'Separation of Duties in Computerized Information Systems', *DBSec*, pp. 179–90.

Tayler, W. B. and R. J. Bloomfield (2011), 'Norms, Conformity, and Controls', *Journal of Accounting Research*, Vol. 49, No. 3, pp. 753–90.

Tirole, J. (1986), 'Hierarchies and Bureaucracies: On the Role of Collusion in Organizations', *Journal of Law, Economics, & Organization*, Vol. 2, No. 2, pp. 181–214.

Van Kesteren, A. and M. Stachowiak (2015), 'HTML Design Principles', in D. B. West, *Introduction to Graph Theory*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Whalen, D. and J. McKeon (2017), *SOX Disclosures: A Thirteen Year Review*, Technical Report, Audit Analytics.

Wu, Z., X. Liu, Z. Ni, D. Yuan, and Y. Yang (2013), 'A Market-oriented Hierarchical Scheduling Strategy in Cloud Workflow Systems', *The Journal of Supercomputing*, pp. 1–38.

Yamalidou, K., J. Moody, M. Lemmon, and P. Antsaklis (1996), 'Feedback Control of Petri Nets Based on Place Invariants, *Automatica*, Vol. 32, No. 1, pp. 15–28.

Zimmermann, H. (1980), 'OSI Reference Model—The ISO Model of Architecture for Open Systems Interconnection', *IEEE Transactions on Communications*, Vol. 28, No. 4, pp. 425–32.

APPENDIX A
MATRIX REPRESENTATIONS AND GRAPH THEORETIC DEFINITIONS

A.1. *Review of Matrix Representations*

As can be seen from the various algorithms presented in this paper, it is advantageous to represent some of the input data in the form of Boolean matrices, that is, matrices where each entry is either 0 or 1. The following Boolean matrices were defined in several subsections of the section titled 'The Setting'. For the reader's convenience, we have summarized the definitions below.

(a) The **Task-Role Matrix**, denoted by TR, has one row corresponding to each task and one column corresponding to each role. An entry $TR[i, j]$ is 1 if task

i requires role j and 0 otherwise. The TR matrix provides an alternative representation of the function f_{TR} and the binary relation $\mathbb{B}$ defined in The Setting'.

(b) The **Role-Person Matrix**, denoted by RP, has one row corresponding to each role and one column corresponding to each person. An entry $RP[j, k]$ is 1 if role j can be played by person k and 0 otherwise. The RP matrix provides an alternative representation of the function f_{PR} defined in The Setting' .

As mentioned earlier, a given assignment α of task-role pairs to persons can also be represented as a matrix. This matrix, which we denote by ALPHA, has the same number of rows and columns as the TR matrix. However, ALPHA is *not* Boolean; its entries take on values from $\{0, 1, 2, \ldots, |P|\}$ with the following convention: if task i does not need role j (that is, $(i,j) \notin \mathbb{B}$), then ALPHA$[i, j] = 0$; otherwise, ALPHA $[i, j]$ stores the value of $\alpha(i, j)$, which is in the range 1 through $|P|$. The advantage of this matrix representation is that given a task i and a role j, the person p assigned to that task-role pair can be found in constant (that is, $O(1)$) time.

A.2. *Relevant Graph Theoretic Definitions*

We give below some definitions and results from graph theory which are used in this paper. These definitions and results can be found in most standard texts on algorithms and graph theory such as Cormen *et al.* (2009) and West (2003).

An undirected graph $G(V, E)$ is **connected** if for each pair of nodes u and v in V, there is a path between u and v in G. When a graph is not connected, it consists of two or more **connected components**; for any two nodes u and v in the same connected component, there is a path between u and v that uses only nodes within that connected component. Several standard algorithms for finding the connected components of an undirected graph $G(V, E)$ are known; these algorithms run in $O(|V| + |E|)$ time (Cormen *et al.*, 2009). We use the idea of connected components in our formal analysis of SoD rules.

A **bipartite** graph $G(V_1, V_2, E)$ is an undirected graph in which the node set is partitioned into two subsets V_1 and V_2 and each edge in E joins a node in V_1 to a node in V_2. Given a bipartite graph $G(V_1, V_2, E)$, a **matching** in G is a subset M of edges such that no two edges of M share an end point. A matching of largest cardinality is called a **maximum matching**. It is well known that for any bipartite graph $G(V_1, V_2, E)$, a maximum matching can be found in $O\left(|E|\sqrt{|V_1| + |V_2|}\right)$ time (Cormen *et al.*, 2009). We use this result in the section titled 'SoD Rule 2: CoI Due to Role Conflicts'.

In some sections, we use directed graphs. In such graphs, a directed edge from node i to node j is denoted by (i, j). We assume that each directed graph $H(V_H, A_H)$ is given by its **adjacency matrix** representation in Cormen *et al.* (2009), which uses a $|V_H| \times |V_H|$ Boolean matrix, say MH, to store the directed edges. An entry MH$[i, j]$ is 1 if the edge (i, j) is in A_H; it is 0 otherwise.

In the section titled 'SoD Rule 4: CoI Due to Common Roles Among Tasks', we also use some known results regarding directed acyclic graphs (dags). Consider

a dag $D(V, A)$ and two nodes u and v in V such that there is a directed path from u to v in D. The **length** of such a path can be measured by either the number of directed edges or the number of nodes. A **longest path** from u to v in D is a path with the maximum number of directed edges (or nodes).

Some of the SoD rules considered in this paper use the precedence dag $D(V, A)$ of the business process. These rules involve pairs of tasks between which there is a directed path in D with one or more edges. (Since D is acyclic, there is no directed path with one or more edges from a task and itself.) Thus, to develop algorithms for those SoD rules, we need to determine whether there is a directed path with one or more edges between a given pair of tasks. In designing and analyzing these algorithms, we assume that through an appropriate preprocessing step, a Boolean $|V| \times |V|$ **reachability matrix** RM has been constructed for D. For any two nodes u and v in D, the RM$[u, v] = 1$ if there is a directed path with one or more edges from u to v and 0 otherwise. Thus, given two nodes u and v in D, the RM matrix enables us to check in $O(1)$ time whether there is a directed path with one or more edges from u to v. It is well known that the necessary preprocessing step (that is, computing the **transitive closure** of D) can be done efficiently (Cormen *et al.*, 2009).

Similarly, we assume that the strict domination relationship among the roles introduced in the second section of the paper is stored in a suitable matrix so that given two roles r_1 and r_2, we can determine in $O(1)$ time whether one of them strictly dominates the other.

APPENDIX B
PROOFS OF SOME THEOREMS

B.1. *Proof of Theorem 2*

Statement of Theorem 2: *Given a business process, the problem of determining whether there is a viable assignment under SoD Rule 1 is* ***NP****-complete even when there are only two types of tasks.*

Proof: We will use Assign-R1 to denote the the problem of determining whether there is a viable assignment under SoD Rule 1.

The Assign-R1 problem is in **NP** since one can guess an assignment α and test in polynomial time (using Algorithm 3) that it is viable under Rule 1.

To prove **NP**-hardness, we use a reduction from the 3SAT problem which is well known to be **NP**-complete (Garey and Johnson (1979)). An instance I of 3SAT consists of a set $X = \{x_1, x_2, \ldots, x_n\}$ of n Boolean variables and a set $Y = \{Y_1, Y_2, \ldots, Y_m\}$ of m clauses, with each clause containing exactly three literals. The question is whether there is an assignment of a Boolean value to each variable in X such that under that assignment, each clause in Y evaluates to `True`. Given an instance I of 3SAT, we construct an instance I' of Assign-R1 as follows.

(a) The business process of I' has only tasks of two types, denoted by `A` and `B`. Each task involves only one role and the roles are all distinct. (Thus, the number of roles is equal to the number of tasks.)

(b) For each variable x_i, we create two people, denoted by $p_{i,\,0}$ and $p_{i,\,1}$, $1 \le i \le n$. Persons $p_{i,\,0}$ and $p_{i,\,1}$ correspond to the literals $\bar{x}_i$ and x_i respectively. Thus, the set $P = \{p_{1,\,0},\ p_{1,\,1}, p_{2,\,0},\ p_{2,\,1},\ \ldots,\ p_{n,\,0},\ p_{n,\,1}\}$ has a total of $2n$ people.

(c) For each variable x_i, we also create a task t_i, $1 \le i \le n$. The type of t_i is **A** and it needs just one role r_i. The role sets of both $p_{i,\,0}$ and $p_{i,\,1}$ include r_i, $1 \le i \le n$. However, r_i is *not* in the role set of any other person in P. Thus, for $1 \le i \le n$, t_i must be assigned to exactly one of $p_{i,\,0}$ and $p_{i,\,1}$.

(d) For each clause Y_j, we create one task w_j and its unique role r_{n+j}, $1 \le j \le m$. The type of task w_j is **B**. The role r_{n+j} appears in the role set of each person whose corresponding literal appears in clause Y_j.

(e) Thus, the set T of $n + m$ tasks is given by $T = \{t_1, t_2, \ldots, t_n, w_1, w_2, \ldots, w_m\}$ and the set R of $n + m$ roles is given by $R = \{r_1, r_2, \ldots, r_{n-1}, r_n, r_{n+1}, \ldots, r_{n+m}\}$.

This completes the construction of the ASSIGN-R1 instance I'. It can be seen that the construction can be carried out in polynomial time. We now show that there is a solution to the ASSIGN-R1 instance I' if and only if there is a solution to the 3SAT instance I. Throughout this proof, the reader should bear in mind that each task in T has a unique role. So, assigning a task to a person p is the same as assigning the corresponding role to p.

Suppose there is a solution to the 3SAT instance I. We construct a solution to the ASSIGN-R1 instance I' as follows. To begin with, all the tasks are unassigned. Consider each variable x_i, $1 \le i \le n$.

(i) If the solution to 3SAT sets x_i to **True**, we assign task t_i (of type **A**) to $p_{i,\,0}$. Also, suppose the positive literal x_i occurs in the set C_i of clauses given by $C_i = \{Y_{j_1}, Y_{j_2}, \ldots, Y_{j_t}\}$. Let $T_i = \{w_{j_1}, w_{j_2}, \ldots, w_{j_t}\}$ be the tasks corresponding to the clauses in C_i. Further, let T_i' denote the subset of T_i such that none of the tasks in T_i' has been assigned to a person. We assign all the tasks in T_i' (each of which is of type **B**) to $p_{i,\,1}$.

(ii) If the solution to 3SAT sets x_i to **False**, we assign task t_i (of type **A**) to $p_{i,\,1}$. Also, suppose the negated literal $\bar{x}_i$ occurs in the set of clauses C_i given by $C_i = \{Y_{j_1}, Y_{j_2}, \ldots, Y_{j_t}\}$. Let $T_i = \{w_{j_1}, w_{j_2}, \ldots, w_{j_t}\}$ be the tasks corresponding to the clauses in C_i. Further, let T_i' denote the subset of T_i such that none of the tasks in T_i' has been assigned to a person. We assign all the tasks in T_i' (each of which is of type **B**) to $p_{i,\,0}$.

It can be verified that this method produces a valid assignment α (that is, α assigns each role r to a person whose role set includes r) that satisfies SoD Rule 1 (that is, all the tasks assigned to each person are of the same type). In other words, we have a solution to the ASSIGN-R1 instance I'.

Now, suppose there is a solution to the ASSIGN-R1 instance I'. We construct a solution to the 3SAT instance I as follows. For each i, $1 \le i \le n$, if task t_i is assigned to $p_{i,\,0}$, we set x_i = **True**; otherwise, we set x_i = **False**. To show that this

gives a solution to the 3SAT instance I, consider any clause Y_j and the corresponding task w_j. Note that w_j is of type **B**. There are two cases to consider.

Case 1: Task w_j is assigned to $p_{i,\,1}$ for some i.

Here, by our construction of instance I', the unnegated literal x_i appears in Y_j. Further, since a task of type **B** has been assigned to $p_{i,\,1}$, task t_i (of type **A**) must be assigned to $p_{i,\,0}$. Hence, our truth assignment step sets x_i to **True**. Since the unnegated literal x_i appears in Y_j, we conclude that Y_j is satisfied.

Case 2: Task w_j is assigned to $p_{i,\,0}$ for some i.

Here, by our construction of instance I', the negated literal $\bar{x_i}$ appears in Y_j. Further, since a task of type **B** has been assigned to $p_{i,\,0}$, task t_i (of type **A**) must be assigned to $p_{i,\,1}$. Hence, our truth assignment step sets x_i to **False**. Since the negated literal $\bar{x_i}$ appears in Y_j, we conclude that Y_j is satisfied.

This completes the proof of Theorem 2. □

B.2. *Proof of Theorem 3*

Statement of Theorem 3: *The following problems can be solved in polynomial time: (i) determining whether a given assignment α is viable under SoD Rule 2 and (ii) given a business process, determining whether there is a viable assignment under SoD Rule 2.*

Proof of Part (i): Our algorithm first checks if α is valid using the algorithm presented in the proof of Part (i) of Lemma 1. If α is valid, it needs to check whether the number of roles assigned to each person is at most one. This can be done efficiently in a manner similar to that presented in Algorithm 3. The idea is to construct an array **W** with $|P|$ elements such that each entry **W**[p] stores the role assigned to person p. We use the convention that if no role has been assigned to p, then **W**[p] = −1. Each entry of the array **W** is initialized to –1. The algorithm goes through each (t, r) pair in $\mathbb{B}$. Suppose ALPHA[t, r] = p. If **W**[p] = −1 (that is, r is the first role assigned to p), it sets **W**[p] to r. Otherwise, it checks whether **W**[p] = r. If it is the case, it consider the next entry of $\mathbb{B}$; otherwise, it report that SoD Rule 2 is not satisfied and stops. Thus, the correctness of the algorithm is obvious.

To estimate the running time, note that from Lemma 1, checking whether α is valid can be done in $O(|\mathbb{B}|)$ time. To check whether α satisfies SoD Rule 2, we note that the time to construct the array **W** is linear in the size of **W** and ALPHA; that is, the time is $O(|P|+|T|\times|R|)$. As can be seen from the above description, the algorithm uses $O(1)$ time for each task-role pair in $\mathbb{B}$; thus, the time to process all the task-role pairs in $\mathbb{B}$ is $O(|\mathbb{B}|)$. So, determining whether each person has been assigned at most one role can be done in $O(|P|+|T|\times|R|+|\mathbb{B}|)$ time, which can be simplified to $O(|P|+|T|\times|R|)$ since $|\mathbb{B}| \le |T| \times |R|$. Since P and ALPHA are all inputs to the problem, the running time of the algorithm is linear in the input size.

FIGURE 7

TR AND RP MATRICES FOR EXAMPLE 9

TR Matrix

	r_1	r_2	r_3	r_4
t_1	1	1	0	0
t_2	0	1	1	0
t_3	0	1	0	1

RP Matrix

	p_1	p_2	p_3	p_4
r_1	1	0	1	1
r_2	1	1	0	0
r_3	1	1	0	1
r_4	0	1	1	0

Note: These were also used in Example 2.

We now present examples to illustrate the algorithm in the above proof for determining whether a given assignment is viable under SoD Rule 2.

EXAMPLE 9

This example uses the same business process as the one used in Example 2. In that example, $T = \{t_1, t_2, t_3\}$, $R = \{r_1, r_2, r_3, r_4\}$ and $P = \{p_1, p_2, p_3, p_4\}$. For the reader's convenience, in Figure 7 we reproduce the task-role (TR) and the role-person (RP) matrices for the business process of Example 2. As before, from the TR matrix, the set $\mathbb{B}$ of task-role pairs is given by

$$\mathbb{B} = \{(t_1,r_1),(t_1,r_2),(t_2,r_2),(t_2,r_3),(t_3,r_2),(t_3,r_4)\}. \tag{1}$$

Let us first consider the following assignment α generated in Example 2.

$$\begin{aligned} &\alpha(t_1,r_1)=p_1 \quad \alpha(t_1,r_2)=p_1 \quad \alpha(t_2,r_2)=p_1 \\ &\alpha(t_2,r_3)=p_2 \quad \alpha(t_3,r_2)=p_2 \quad \alpha(t_3,r_4)=p_3 \end{aligned} \tag{2}$$

As mentioned in the discussion for Example 2, the above assignment α is valid. For an assignment to be viable under SoD Rule 2, it must be valid and each person must be assigned at most one role. In the assignment α, we note that p_1 has two roles, namely r_1 and r_2. Hence, our algorithm would report that the above assignment is not viable under SoD Rule 2.

EXAMPLE 10

For the business process in Example 9, let us consider the following assignment α_1 (instead of the assignment α used in Example 9).

$$\begin{array}{lll} \alpha_1(t_1,r_1)=p_1 & \alpha_1(t_1,r_2)=p_2 & \alpha_1(t_2,r_2)=p_2 \\ \alpha_1(t_2,r_3)=p_4 & \alpha_1(t_3,r_2)=p_2 & \alpha_1(t_3,r_4)=p_3 \end{array} \tag{3}$$

Using the RP matrix, it can be seen that the assignment is valid. Further, each person is assigned is exactly one role: the roles assigned to p_1, p_2, p_3 and p_4 are r_1, r_2, r_4 and r_3 respectively. Thus, our algorithm would report that α_1 is a viable assignment under SoD Rule 2.

Proof of Part (ii): Our procedure for testing whether there is a viable assignment under SoD Rule 2 is shown in Algorithm 11. (It uses the notion of matching in bipartite graphs introduced in Section A.2.) We will now argue the correctness of the algorithm and explain how to construct such a viable assignment when the algorithm reports 'Yes'.

Algorithm 11. Finding a viable assignment under SoD Rule 2

```
1  Construct the bipartite graph G(V_P, V_R, E), where the node sets V_P and V_R
     are in one-to-one correspondence with sets P (the set of people) and R (the
     set of roles) respectively. For each node p ∈ V_P and r ∈ V_R, the edge {p, r} is
     in E if and only if the role set of the person represented by p includes the
     role represented by r.
2  Find a maximum matching M in G.
3  if (| M| ≠ | R| ) then
4  |  output "There is no viable assignment under SoD Rule 2".
5  else
6  |  for each edge {p, r} in M do
7  |  |  Assign role r in all the tasks that need r to person p.
8  |  end
9  |  output the assignment.
10 end
```

Suppose the algorithm outputs 'Yes'. The matching M with $|R|$ edges gives an assignment of at most one role to each person as follows: for each edge $\{p, r\} \in M$, the role represented by r is assigned to the person represented by p in every task that needs role r. (This is indicated in the loop in Step 6 of Algorithm 11.) By the definition of G, the presence of the edge $\{p, r\}$ ensures that the person represented by p can play the role represented by r. Further, the fact that the matching has $|R|$

edges ensures that each role is assigned to some person. Hence, we have a viable assignment under SoD Rule 2.

For the converse, suppose there is a viable assignment under SoD Rule 1; that is, each role is assigned to some person and each person is assigned at most one role (across all tasks). It can be seen that the assignment corresponds to a matching with $|R|$ edges in G. Thus, the algorithm would output 'Yes'.

To estimate the running time, we note that in Step 1 of Algorithm 11, the graph G can be constructed using the RP matrix in $O(|P| \times |R|)$ time. Since G has $|P| + |R|$ nodes and at most $|P| \times |R|$ edges, finding a maximum matching in G (Step 2) can be done in $O\left(|P| \times |R| \times \sqrt{|P| + |R|}\right)$ time as mentioned in Section A.2. To construct the assignment (when one exists), the number of iterations of the loop in Step 6 is $|M| = |R|$. For each role r, we can find the set of all tasks that need role r in $O(|T|)$ time using the TR matrix. Therefore, finding a viable assignment can be done in $O(|R| \times |T|)$ time. Therefore, the overall running time is $O\left(|P| \times |R| \times \sqrt{|P| + |R|} + |R| \times |T|\right)$. □