Simulation

Simulation: Transactions of the Society for Modeling and Simulation International
88(7) 852–869
© The Author(s) 2011
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/0037549711417880
sim.sagepub.com

A framework for simulation-based optimization of business process models

Farzad Kamrani[1], Rassul Ayani[1] and Farshad Moradi[2]

Abstract

The Assignment Problem is a classical problem in the field of combinatorial optimization, having a wide range of applications in a variety of contexts. In general terms, the Assignment Problem consists of determining the best assignment of tasks to agents according to a predefined objective function. Different variants of the Assignment Problem have been extensively investigated in the literature in the last 50 years. In this work, we introduce and analyze the problem of optimizing a business process model with the objective of finding the most beneficial assignment of tasks to agents. Despite similarities, this problem is distinguished from the traditional Assignment Problem in that we consider tasks to be part of a business process model, being interconnected according to defined rules and constraints. In other words, assigning a business process to agents is a more complex form of the Assignment Problem. Two main categories of business processes, *assignment-independent* and *assignment-dependent*, are distinguished. In the first category, different assignments of tasks to agents do not affect the flow of the business process, while processes in the second category contain critical tasks that may change the workflow, depending on who performs them. In each category several types of processes are studied. Algorithms for finding optimal and near-optimal solutions to these categories are presented. For the first category, depending on the type of process, the Hungarian algorithm is combined with either the analytical method or simulation to provide an optimal solution. For the second category, we introduce two algorithms. The first one finds an optimal solution, but is feasible only when the number of critical tasks is small. The second algorithm is applicable to large number of critical tasks, but provides a near-optimal solution. In the second algorithm a hill-climbing heuristic method is combined with the Hungarian algorithm and simulation to find an overall near-optimal solution. A series of tests is conducted which demonstrates that the proposed algorithms efficiently find optimal solutions for assignment-independent and near-optimal solutions for assignment-dependent processes.

Keywords

Assignment Problem, business process optimization, Hungarian algorithm, simulation-based optimization

1. Introduction

The demanding and complex task of improving the performance of a business process may be viewed from various perspectives. Different disciplines have provided a vast range of approaches to the problem, which have been employed by business and other types of organizations over the years with varying degrees of success. Among examples of suggested measures that may improve a business process, the following can be highlighted: (1) increasing the quality and performance of the workforce by education and training; (2) improving the structure of an organization and work process; (3) improving the quality of the management of organization; and (4) automation and introducing technical aids that can enhance the performance of the personnel. Common for these solutions is that they are more or less long-term measures, which are complex and costly to implement and may require changes to the organization and the business process.

[1]School of Information and Communication Technology, KTH Royal Institute of Technology, Sweden.
[2]Division of Information Systems, FOI, Swedish Defence Research Agency, Sweden.

Corresponding author:
Farzad Kamrani, School of Information and Communication Technology, KTH Royal Institute of Technology, Stockholm, Sweden
Email: kamrani@kth.se

Nevertheless, another equally significant approach, which is the focus of this paper, is to improve the performance of a business process by selecting the most qualified available personnel for each task, without refining the structure of the business process or hiring new employees.

This approach is appropriate for scenarios in which a given number of personnel within an organization are dedicated to perform a defined activity. For instance, consider military personnel who are appointed to positions in a command center, or a group of software engineers in a company that are assigned the task of developing a software system. These types of business processes consist of different tasks, which require various categories of expertise. Appointed personnel are usually in varying degrees qualified for the involved activities. However, since their performance may vary for different tasks, the outcome of the process depends heavily on which tasks are assigned to which employees.

The problem of optimally assigning tasks to workers, which is known as the *Assignment Problem (AP)*, is not new and different variants of it have been discussed for more than 50 years, that is, since the introduction of the first algorithm that solved it in polynomial time. Pentico[1] provides a valuable survey over variations of the AP. Despite similarities, these problems have varying degrees of complexity. While the optimal solution to some of them, such as the classical AP, has a polynomial time complexity, some others (such as the *Generalized AP*) are non-deterministic polynomial-time (NP)-hard combinatorial optimization problems[2] and are usually solved by approximate methods.

Approximate methods do not guarantee an optimal solution. However, they usually find high-quality solutions in a significantly reduced amount of time. A very popular group of approximate methods, commonly called metaheuristics, try to combine basic heuristic methods in higher level frameworks to efficiently explore a search space. Metaheuristic algorithms are non-deterministic strategies that ‘guide’ the search process in order to find near-optimal solutions. They usually incorporate mechanisms to avoid getting trapped in confined areas of the search space.[3] Metaheuristics include – but are not restricted to – the *Genetic Algorithm (GA)*, *Iterated Local Search (ILS)*, *Simulated Annealing (SA)* and *Tabu Search (TS)*.

The GA[4,5] is a metaheuristic based on evolutionary ideas of natural selection and genetics, which was invented by John Holland.[6] It simulates the ‘survival of the fittest’ principle that was laid down by Charles Darwin. The GA has frequently been used for solving many optimization problems.[7–10]

The ILS and TS are two different local search metaheuristics. In the ILS one generates an initial solution and finds a local optimum by performing a local search. Then, the obtained solution is modified randomly and the search continues until a new solution is reached. This procedure is repeated iteratively until a termination criterion is met. The TS is also an iterative local search. The main feature of the TS is its use of an adaptive memory that prevents the algorithm of revisiting solutions, which have already been explored.

SA is a metaheuristic mainly used in combinatorial optimization, which is inspired by the basic principles of statistical thermodynamics and simulates the transition to the equilibrium and decreasing the temperature to find smaller and smaller values of the mean energy of a system. In each step of the SA algorithm the current solution is replaced by a random solution from its neighborhood. The probability that a new solution is accepted depends on the difference between the corresponding objective values and a global parameter T, called the temperature. Large values of T correspond to increasing randomness and, as T goes toward zero, solutions with more differences in objective values are selected. One should gradually decrease the temperature during the process.[11,12]

Despite similarities, our work is distinguished from the AP in that we consider tasks as part of a business process, being interconnected to each other according to given business rules. The process usually includes decision points at which the path of the workflow is determined. Depending on different factors, some tasks may be repeated several times or different courses of action (COAs) may be chosen resulting in alternative tasks to be performed (see Figure 1). A business process often involves uncertainty that must be addressed and incorporated into the analysis in an adequate manner. One can say that assigning a business process to personnel is a more complex form of the AP.

Depending on the type of the business process model, different solutions are required. Our approach has been to find the optimal solution employing exact methods, and avoid using heuristics to the extent that it is possible. Even in the most general case where no exact method exists, we do not rely entirely on heuristics and combine them with exact methods used for different partitions of the problem.

The outline of the rest of this paper is as follows. In Section 2, the business process modeling and our choice of modeling tool are discussed. In Section 3, the problem formulation and the objective function are presented. In Section 4, we suggest a model for estimating the performance of personnel based on their capabilities. In Section 5, several simulation-based algorithms for finding the optimal and near-optimal solutions are presented. In Section 6, the computational complexity of the methods is investigated. Section 7 discusses results from test scenarios. In

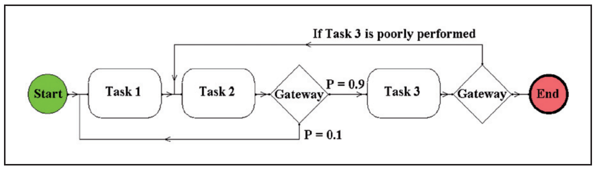


Figure 1. A process with three tasks. Gateways after tasks 2 and 3 may change the workflow.

Section 8, some restrictions of our approach are addressed. Section 9 concludes the paper and discusses future work.

2. Business process modeling

Business process modeling refers to describing business processes at a high abstraction level, by means of a formal notation to represent activities and their causal and temporal relationships, as well as specific business rules that process executions have to comply with.[13] The focus of business process modeling is the representation of the execution order of activities, which is described through constructs for sequence, choice, parallelism or synchronization.[14] Over the years, various business modeling methodologies, such as the *flow chart*, *data flow diagram*, *control flow diagram*, *Gantt chart* and *Unified Modeling Language (UML)*, have been employed to model and analyze different aspects of business processes. In recent years, *Business Process Modeling Notation (BPMN)*[15] has emerged as the de facto standard for modeling organizations and business processes.[16–18] In this paper, we choose BPMN as the modeling tool of business processes, due to its intuitional appeal and wide acceptance.

BPMN is a graphical notation that provides organizations with the capability to document, analyze and communicate their business procedures in a simple and standard manner. The graphical notation of BPMN is chosen to resemble the flow-chart format. One of the objectives in the development of BPMN has been to create a simple and understandable mechanism for creating business process models, while at the same time being able to handle the complexity inherent in business processes. For this reason, the graphical notation is organized into a small set of specific notation categories. While the reader of a BPMN diagram can easily recognize the basic types of elements and understand the diagram, additional variations and information are added within the basic elements to support the requirements for complexity without dramatically changing the basic look and feel of the diagram.[15]

In BPMN, the association of a particular action or set of actions with a specific resource is illustrated through the use of the *Pool* and *Lane* constructs, commonly called *Swimlanes*. However, Wohed et al.[19] show that the BPMN's support for the resource perspective is not sufficient. To overcome this limitation, an extension using UML constructs is introduced by Siegeris and Grasl[20] and an example of process modeling for a large-scale modeling effort using BPMN is provided.

Even though one of the major goals of business process modeling and BPMN is to provide a means to understand and analyze the business structure, it is possible to extend BPMN to cover other aspects of a process model. For example, Magnani and Montesi[21] extend BPMN to evaluate the (monetary) cost of a business process. They suggest two methods: *cost intervals*, in which the cost of a task is expressed by its lower and upper limits, and *average cost*, in which an average cost is assigned to each task together with the probability of alternative paths. In both methods costs are aggregated in an additive manner to the overall cost of the entire process.

Similar to this approach, we suggest extending the BPMN constructs, such as *Task* and *Activity*, with a performance metric *value added*, which describes how much the final value of the outcome of the business process will increase as the workflow proceeds through each BPMN element. The *total value added* by a process then is the sum of the value added in each step of the business process.

Evidently, measuring the value added by each stage in a business process is very challenging, especially when human performers are part of the process. The versatility of individual characteristics and the unpredictable nature of human beings make any estimation of the value of their work effort rather speculative. Individual characteristics, such as cognitive ability, motivation, mental models, expertise, experience, creativity and mood, all have critical impacts on the performance of individuals. Nevertheless, we suggest that the value added in a business process is the result of the qualified personnel's work effort, and a business process in which qualified personnel are working with tasks that require these qualifications produces an output that is more thoroughly worked out. In other words, we assume that more qualified work results in higher value added to the outcome of a business process. With qualified work, we mean work efforts that are required for completion of a process. For instance, in a software development task that requires only C-programming skills, a higher amount of work efforts by personnel who lack this capability produces no added value.

The BPMN extension that we propose and use in this paper is the following. The performers of Tasks and Activities of a business process are modeled as a collection of *m agents*, $\{a_1, \ldots, a_m\}$. Each agent, a_i, has a set of *n* attributes $c_i = \{c_{i1}, \ldots, c_{in}\}$, which are numerical values that quantify its capabilities. This set, that we from now on call the *capability vector*, defines all possible qualifications that may be of interest for the entire business process. For instance, in a software development process, various programming skills may be examples of capabilities of agents, or in a military planning team, more subjective characteristics, such as cognitive ability, creativity and communication skills, may constitute the set of the required capabilities. Capabilities are graded by a number ranging between 0 and 5, where 0 is lack of the capability and 5 is the highest possible value. In many cases, these grades may be based on objective measures; for example, experience can be measured by number of years in a business, or technical skills may be graded by performance in a training course. In other cases the grades may be subjective values assigned by the decision maker; for instance, the group leader may grade agents according to their earlier achievements, based on his own experience.

For each task, t_l, a weight is associated to each capability. These values define the (capability) *weight vector*, that is, $w_l = \{w_{1l}, \dots, w_{nl}\}$. Each value, w_{jl}, describes the importance of the capability j for task l. These values are assigned by domain experts and normalized so that the sum of weights for each task is equal to 1, that is, $\sum_{j=1}^{n} w_{jl} = 1, \quad \forall t_l$.

3. Problem formulation

In Kamrani et al.,[22] we proposed a model for estimating the value added by human agents in a business process. Here, we present a modified version of the same model, which incorporates some more details. In this model the value added by an agent to a task (per time unit) is described as a function of the capabilities of the agent and the importance of these capabilities for the task:

$$v_{il} = f(c_i, w_l), \tag{1}$$

where v_{il} is the value added (per time unit) by agent a_i in performing task t_l, $c_i = \{c_{i1}, \dots, c_{in}\}$ is the capability vector of agent a_i and $w_l = \{w_{1l}, \dots, w_{nl}\}$ is the weight vector of task t_l.

In a business process model, some tasks are performed a number of times. This number may be predetermined or modeled as a random variable with a distribution that is either fixed or depends on the quality of the performance of the current or other tasks. For instance, consider the process shown in Figure 1. The probability that after task 2 the workflow continues to task 3 is always 0.9. However, the process is completed only if task 3 is sufficiently well performed, otherwise the workflow is directed to task 2. Generally, this measure depends on which agent has performed task 3.

In aggregating values added by agents to a process, the following factors should be considered.

1. How many times each task is performed – we denote this number by x_l for task l.
2. The assignment of tasks to agents. We denote this scheme by the assignment matrix $Z = [z_{il}]_{m \times s}$, where m and s denote numbers of agents and tasks, respectively, and $z_{il} = 1$ if task l is assigned to agent i and 0 otherwise.

If the time required to carry out task t_l is denoted by q_l and it is performed x_l times before the process is finished, the total value added to the process is

$$v = \sum_{l=1}^{s} \sum_{i=1}^{m} x_l q_l f(c_i, w_l) z_{il}. \tag{2}$$

However, the value v is a gross measure, since it does not take into account the costs of agents. In order to obtain the overall measure of performance, we consider the costs and subtract the total cost of agents from the added value. We assume that the cost of an agent (per time unit) is a function of its capabilities and independent of the task it is performing, that is, the cost of agent a_i (per time unit) is $e_i = g(c_i)$, where $c_i = \{c_{i1}, \dots, c_{in}\}$ is the set of the agent's capabilities. Then the total cost of all agents performing the business process will be

$$e = \sum_{l=1}^{s} \sum_{i=1}^{m} x_l q_l g(c_i) z_{il}, \tag{3}$$

where x_l, q_l and z_{il} are defined as above for Equation (2). By subtracting Equation (3) from Equation (2), the objective function of the business process:

$$u = v - e = \sum_{l=1}^{s} \sum_{i=1}^{m} x_l q_l \left(f(c_i, w_l) - g(c_i) \right) z_{il} \tag{4}$$

is obtained. Given functions f and g, Equation (4) yields a single measure of performance (objective function) for the business process by reducing the costs of agents from the sum of values added by them. Thus, it is desirable to maximize this objective function, subject to any constraints on the number of agents and tasks.

We define the value *added matrix* as

$$V = [v_{il}]_{m \times s}, \tag{5}$$

where $v_{il} = f(c_i, w_l)$, and the agent *cost vector* as

$$COST = [cost_i]_{m \times 1}, \tag{6}$$

where $cost_i = g(c_i)$. Functions f and g in Equation (4) can be substituted by v_{il} and $cost_i$ in order to obtain a more convenient form of the equation.

Then, the optimization problem can be formulated as the following. Given the objective function

$$u = \sum_{l=1}^{s} \sum_{i=1}^{m} x_l q_l \, (v_{il} - cost_i) z_{il} \tag{7}$$

find an assignment matrix Z, that is, all $z_{il} \in \{0, 1\}$, such that u is maximized, subject to constraints

$$\sum_{i=1}^{m} z_{il} = 1, \text{ for all } l \in \{1, \dots s\} \tag{8}$$

$$\sum_{l=1}^{s} z_{il} = 1, \text{ for all } i \in \{1, \dots m\} \tag{9}$$

Constraints (8) and (9) mean that each task is assigned to one agent and each agent performs one task, implying that numbers of tasks and agents are equal. If the number of tasks and agents are not equal, one can balance them by introducing ‘dummy’ tasks or agents, which do not add any value to the process. In the final solution, those tasks that are assigned to dummy agents (or vice versa) remain unassigned.

In a more generalized form of the problem, a task may require several agents, that is constraint (8) is replaced by

$$\sum_{i=1}^{m} z_{il} = d_l, \text{ for all } l \in \{1, \dots s\} \tag{10}$$

where d_l is the number of agents performing task t_l and $\sum_{l=1}^{s} d_l = m$. However, this problem can also be reformulated as the original one with an equal number of tasks and agents, by decomposing each task t_l to d_l tasks, each of which require one agent. Therefore, without any loss of generality, we may assume that the number of tasks and agents are equal and constraints (8) and (9) are applied. However, we presuppose that each agent is involved in only one task and agents performing a task do not interact with each other.

The main difficulty in solving the stated optimization problem, despite its similarity with the classical AP, is that the variable x_l generally depends on the assignment matrix Z, as will be discussed in Section 5.

4. Approximate models

In this section, two models, which supply the value added matrix (V) and the agent cost vector ($COST$), are presented. We assume that these values are expressed in appropriate units, which are consistent with the grading system and capability weights suggested earlier and no scaling factor is required to convert between units.

For each agent a_i and task t_l, we approximate v_{il} by the sum of capabilities of the agent weighted by importance of the capabilities, that is

$$v_{il} = \sum_{j=1}^{n} c_{ij} w_{jl}. \tag{11}$$

The above can be expressed in matrix form in order to simplify the calculations. Considering m agents each having n attributes, the *capability matrix* is defined as $C = [c_{ij}]_{m \times n}$, where c_{ij} is the attribute j of agent i. In a similar way, given s tasks the *weight matrix* is defined as $W = [w_{jl}]_{n \times s}$, where w_{jl} is the weight of attribute j for task l. Then, the value added matrix $V = [v_{il}]_{m \times s}$, where v_{il} is the value added by agent i to task l (per time unit) is calculated by

$$V = CW. \tag{12}$$

The cost of an agent ($cost_i \in COST$) is approximated by a step function of the sum of its capabilities. This model appears to be simple yet realistic. In an organization usually the salaries of the employees are decided by their qualifications and are not changed with the details of the task that they are working with. We assume the cost of each agent (per time unit) is one of three constant values, depending on its capabilities:

$$cost_i = \begin{cases} k_1, & \sum_{j=1}^{n} c_{ij} < \text{normal} \\ k_2, & \text{normal} \le \sum_{j=1}^{n} c_{ij} < \text{expert} \\ k_3, & \sum_{j=1}^{n} c_{ij} \ge \text{expert.} \end{cases} \tag{13}$$

Equations (11) and (13) are incorporated in Equation (7) in order to completely define the objective function.

5. Proposed solution

In its most general case, the objective function defined in Equation (7) incorporates a random element, $x_l\,(Z)$, which depends on the assignment matrix, and there is no easy way to find an optimal solution. However, depending on the type of process, the problem may be reduced to simpler cases, which require less effort to solve. We distinguish between two main categories of processes: (1) those for which the workflow is independent of the assignment of tasks; and (2) those for which depending on the assignment of tasks the flow of the process may be changed. In both *assignment-independent* and *assignment-dependent* categories, the process may be *deterministic*, *Markovian* or *non-Markovian*.

In the following, we discuss first the assignment-independent category, which is less complicated, present solutions to each case and finally introduce an algorithm for the more general cases.

5.1. Deterministic processes

When no randomness is involved in the process and the value of x_l for all t_l is predetermined and independent of Z, the problem is reduced to the standard AP. For each task t_l, the value x_l can be read directly from the business process model. Equation (7) can be rewritten as $u = \sum_{l=1}^{s} \sum_{i=1}^{m} v'_{il} z_{il}$, where $v'_{il} = x_l q_l (v_{il} - cost_i)$. Thus the goal of the optimization problem will be to maximize u subject to constraints (8) and (9).

The number of feasible solutions to the AP grows factorially with the number of tasks (agents), which makes an exhaustive search method practically impossible. However, the well-known Hungarian algorithm solves the AP in polynomial time of the number of tasks. The Hungarian algorithm, also known as the Kuhn–Munkres algorithm, was originally developed and published by Harold Kuhn[23] and had a fundamental influence on combinatorial optimization.[24] Since we adopt this method in our algorithms presented later, we outline the key steps of the method here.

The matrix interpretation of the Hungarian algorithm assumes a non-negative matrix $C = [cost_{il}]_{m\times m}$, where element $cost_{il}$ represents the cost of assigning task l to agent i. The aim of the algorithm is to select m *independent* elements of matrix C such that the total assignment cost is minimized. A set of elements of a matrix are considered to be independent if no two of them are in the same row or column.

A problem in which the objective is to maximize the profit can be easily converted into a minimization problem. Assume that the profit of assigning tasks to agents is defined by a *profit matrix*, $P = [p_{il}]_{m\times m}$. By replacing each p_{il} with $p_{max} - p_{il}$, where p_{max} is the largest element of P, a 'cost' matrix is constructed. An assignment that minimizes the total 'cost' maximizes the profit for the original problem.

The Hungarian algorithm is based on the property that the solution of the problem is not changed if we add a constant to every element of a row (or column) in the cost matrix. Obviously, the total cost is increased by the same constant. It also relies on *König's theorem*, which states that the maximum number of independent zero elements in a square matrix is equal to the minimum number of lines (through a row or a column) required to cover all zero elements.[25] The algorithm consists of the following steps.[25,26]

1. Subtract the smallest element in each row from every element of that row.
2. Subtract the smallest element in each column from every element of that column. After these two steps, the obtained matrix (C_1) contains at least one zero element in each row and each column and has the same solution as the original matrix C.
3. Cover all zero elements in matrix C_1 by drawing as few lines as possible through rows and columns. Let m_1 denote the number of these lines. According to König's theorem, m_1 is equal to the maximum number of independent zero elements. If m_1 is equal to the number of tasks (m), a set of m independent zero elements in C_1 can be identified. Since elements of C_1 are non-negative and the total cost of assignments in the selected set is zero, an optimal assignment for cost matrix C_1 is found. This assignment constitutes the optimal solution for C. The total cost is found by adding the corresponding elements of C in the same positions.
4. If $m_1 < m$, denote the smallest uncovered element of C_1 by h. Add h to every element of each covered row or column of C_1. This implies that h is added twice to each twice-covered element. Subtract h from every element of the matrix. Call the new matrix C_2. This matrix has the same optimal solution as C_1 and C. However, matrix C_2 contains at least one new zero element and the sum of its elements is decreased by $m(m - m_1)h$.
5. Repeat steps 3 and 4, using C_2 instead of C_1.

We refer the interested reader to Munkres,[25] Sasieni et al.[26] and Jonker and Volgenant[27] for a detailed procedure for carrying out step 3, without which the algorithm is still not complete. The time complexity of the Hungarian algorithm is $O(m^3)$, where m is the number of tasks.

We employ the Hungarian algorithm as a function that takes a matrix $V = [v_{il}]_{m\times s}$ as input and returns an optimal assignment matrix Z. This function is denoted as *hungarian* in the following algorithms. The steps of finding the optimal solution to the deterministic (assignment-independent) process are summarized in Algorithm 1.

Algorithm 1. *Optimal assignment of a deterministic process.*

```
given: agents A, tasks T, business process model BPM
return: optimal assignment Z
1  $V \leftarrow CW$ (Equation (12))
2  $COST \leftarrow$ (Equation (13))
3  $X \leftarrow BPM$
4  for each agent $a_i \in A$
5     for each task $t_l \in T$
6        $v'_{il} \leftarrow x_l q_l (v_{il} - cost_i)$
7     end for
8  end for
9  $Z \leftarrow hungarian\ ([v'_{il}]_{m \times s})$
```

5.2. Markovian processes

In many business processes the workflow is not predetermined and may take different paths. Thus, the value of the objective function u as defined by Equation (7) is a random variable, and a natural approach is to find an assignment scheme that maximizes the expected value $E[u]$.

If the uncertainty in the business process is modeled by fixed probabilities assigned to alternative paths, then the business process model can be considered as a *Markov chain*. BPMN's *Tasks* will constitute transient states and *End Events* will be absorbing states of the Markov chain. Using Equation (7), the expected value of the objective function is calculated by

$$E[u] = \sum_{l=1}^{s} \sum_{i=1}^{m} E[x_l] q_l (v_{il} - cost_i) z_{il}. \tag{14}$$

It can be shown that $E[x_l]$ is the first row of the invertible matrix $(I - Q)^{-1}$. Here, I is the identity matrix and Q is obtained from the transition matrix of the Markov chain if we strike off all rows and columns containing absorbing states.[22] In other words, the expected number of times each task is performed can be calculated by

$$E[X] = (1,\ 0,\ 0,\ \cdots)(I - Q)^{-1}, \tag{15}$$

Where $X = [x_l]_{1 \times s} = [x_1,\ x_2,\ \ldots,\ x_s]$ is the number of times each task is performed.

Using Equations (14) and (15), the optimal assignment is found by the Hungarian algorithm. These steps are summarized in Algorithm 2.

Algorithm 2. *Optimal assignment of a Markovian process.*

```
given: agents A, tasks T, business process model BPM
return: optimal assignment Z
1   $V \leftarrow CW$ (Equation (12))
2   $COST \leftarrow$ (Equation (13))
3   derive Markov chain M from BPM
4   $E[X] \leftarrow (1,\ 0,\ 0,\ \ldots)(I - Q)^{-1}$ (Equation (15))
5   for each agent $a_i \in A$
6      for each task $t_l \in T$
7         $v'_{il} \leftarrow E[x_l] q_l (v_{il} - cost_i)$
8      end for
9   end for
10  $Z \leftarrow hungarian\ ([v'_{il}]_{m \times s})$
```

5.3. Non-Markovian processes

Although the presented analytical method in Section 5.2 is appealing, it can only be used for a Markovian process. In modeling a real-world business process, the Markovian constraint is usually too restrictive and not justified. The reason is that the probability that the workflow takes a path generally depends not only on the current state of the process, but also on the history of the workflow. For non-Markovian processes there is no analytical solution and alternative approaches, such as simulation, must be considered.

The simulation method follows the intuitive structure of the BPMN diagram. Tokens are generated at *Start Events* and are propagated along *Sequence Flows*, across *Tasks*, *Activities* and *Gateways*, being duplicated and merged when necessary, until they are consumed by an *End Event*. An element holding a token is considered to be active, which may result in updating the values of some parameters. The simulation is repeated a sufficient number of times so that the average values of the desired parameters are calculated. The aim of the simulation is to estimate $E[X]$, that is, the expected number of times each task is performed. Once $E[X]$ is estimated, the objective function can be calculated as in Equation (14) and the Hungarian algorithm can be adopted to efficiently find the optimal solution. The approach is shown in Algorithm 3.

Algorithm 3. Optimal assignment of a non-Markovian process.

```
given: agents A, tasks T, business process model BPM
return: optimal assignment Z

1   $V \leftarrow CW$ (Equation (12))
2   $COST \leftarrow$ (Equation (13))
3   run simulation of the BPM
4   $E[X] \leftarrow$ average token through each task
5   for each agent $a_i \in A$
6     for each task $t_l \in T$
7       $v'_{il} \leftarrow E[x_l]q_l(v_{il} - cost_i)$
8     end for
9   end for
10  $Z \leftarrow hungarian\ ([v'_{il}]_{m\times s})$
```

The observant reader may have noticed the similarities between Algorithms 1, 2 and 3. Indeed, they are identical except for lines 3–6 of Algorithm 1, and lines 3–7 of Algorithms 2 and 3. Using different methods, this part of the algorithms calculate or estimate the number of times each task is performed in order to calculate the *profit matrix*, $V' = [v'_{il}]_{m\times s}$. Since the discussed processes are independent of the assignment matrix, regardless of the assignments this matrix will be the same and the Hungarian algorithm can be used to efficiently find the optimal solution.

5.4. Assignment-dependent processes

In all three cases discussed above, that is, deterministic, Markovian and non-Markovian processes, we assumed that there is no correlation between the assignment scheme and the path of the workflow or on the probabilities of branches at decision points. However, in many situations this is not the case. For instance, assignment of a task to a less qualified agent may increase the probability that the task is repeated several times. In such scenarios some assignment combinations may change the probabilities that govern the path of the workflow. This means that x_l in Equation (7) is a function of the assignment matrix Z. The algorithms as presented above do not result in a unique matrix V' and different assignments may result in different matrices.

We distinguish between tasks whose assignment may affect the flow of the process and other tasks. From now on, we call such tasks *critical tasks*, and denote the set of these tasks by T_c. If the set of *non-critical tasks* is denoted by T_{nc}, we have $T_c \cup T_{nc} = T$ and $T_c \cap T_{nc} = \phi$. For instance, in Figure 1, task 3 is critical, whereas tasks 1 and 2 are non-critical. Moreover, we denote the assignment of critical and non-critical tasks as Z_c and Z_{nc}, respectively. We use $BPM(Z_c)$ to denote a business process model BPM, whose critical tasks are assigned by Z_c. Given $BPM(Z_c)$, it is possible to calculate or estimate the number of times each non-critical task is performed and determine the profit matrix for the specific assignment of critical tasks.

The function *profit_matrix*, which is described in Algorithm 4, shows the details of this algorithm. This function is also used by algorithms presented later in this section.

Algorithm 4. *Building profit matrix.*

```
given: agents A, tasks T, business process model BPM
       critical task $T_c$, critical assignments $Z_c$
return: matrix V'

1   $V \leftarrow CW$ (Equation (12))
2   $COST \leftarrow$ (Equation (13))
3   assign critical tasks to agents according to $Z_c$
4   if $BPM(Z_c)$ deterministic
5     $X \leftarrow BPM$
6   else if $BPM(Z_c)$ Markovian
7     derive Markov chain M from $BPM(Z_c)$
8     $X \leftarrow (1, 0, 0, \ldots)(I - Q)^{-1}$ (Equation (15))
9   else if $BPM(Z_c)$ non-Markovian
10    run simulation of the $BPM(Z_c)$
11    $X \leftarrow$ average token through each task
12  end if
13  for each agent $a_i \in A$
14    for each task $t_l \in T$
15      $v'_{il} \leftarrow x_l q_l (v_{il} - cost_i)$
16    end for
17  end for
18  return $V' = [v'_{il}]_{m\times s}$
```

For the optimization of assignment-dependent processes, we present two types of solutions. The first one, which is given in Algorithm 5, yields the optimal assignment but is only feasible for scenarios where the number of critical tasks is small. The second solution, shown in Algorithm 6, finds near-optimal solutions in the general case. The input of both of these algorithms can be a deterministic, Markovian or non-Markovian process.

Algorithm 5: *Optimal Assignment of a Dependent Process*

given: *agents A, tasks T, business process model BPM*
return: *optimal assignment Z*

```
1   $T_c \leftarrow BPM$
2   $\tau_c \leftarrow$ number of tasks in $T_c$
3   $\Omega \leftarrow$ all possible permutations of $\tau_c$ agents $\in A$
4   $max\_gain \leftarrow -\infty$
5   for each $\omega \in \Omega$
6     $Z_c \leftarrow$ assign tasks in $T_c$ to agents in $\omega$
7     $V' \leftarrow$ profit_matrix $(A, T, BPM, T_c, Z_c)$
8     $Z_{nc} \leftarrow$ hungarian $(V')$
9     $gain \leftarrow u(Z_c \cup Z_{nc})$
10    if $gain > max\_gain$
11      $max\_gain \leftarrow gain$
12      $Z \leftarrow Z_c \cup Z_{nc}$
13    end if
14  end for
```

In Algorithm 5, first the set of critical tasks is determined (line 1). Naming the number of critical tasks τ_c (line 2), the set (Ω) of all ordered sequences of τ_c distinct agents is created (line 3). For each permutation, critical tasks are assigned to agents in ω (line 6). The profit matrix *V'* for the current assignment is determined by calling the *profit_matrix* function (line 7) and the Hungarian algorithm is applied on this matrix (line 8). Considering both critical and non-critical assignments, the total profit of the business process model is calculated (line 9), and if this value is higher than earlier obtained results (line 10), the answer is updated (line 12).

The basic idea of the algorithm is that by partitioning the tasks into two parts and by employing the Hungarian algorithm to find the optimal assignment for non-critical tasks, the run-time complexity of the algorithm is kept relatively low. The for-loop (lines 5–14) in the algorithm iterates $(\tau_c + \tau_{nc})!/\tau_{nc}!$ times, where τ_{nc} and τ_c are the cardinalities of T_{nc} and T_c, respectively. In each iteration, the two most time-consuming operations are calling the *profit_matrix* function (line 7) and the *hungarian* function (line 8). The Hungarian algorithm is $O(\tau_{nc}^3)$ and the *profit_matrix* function is generally linear in the number of tasks, as will be discussed in the next section. Thus, the total complexity of the algorithm is $O((\tau_c + \tau_{nc})!\tau_{nc}^3/\tau_{nc}!)$. Clearly, this algorithm is computationally feasible only for small values of τ_c.

A polynomial algorithm that finds near-optimal solution for deterministic, Markovian and non-Markovian (assignment-dependent) processes is presented in Algorithm 6. This algorithm also relies on

Algorithm 6: Near-optimal Assignment of a Dependent Process

given: *agents A, tasks T, business process model BPM*
return: *near-optimal assignment Z*

```
1   $T_c \leftarrow BPM$
2   $T_{nc} \leftarrow T \setminus T_c$
3   $Z_c \leftarrow$ assign $T_c$ to a random feasible set of agents
        $A_b$ and mark all agents $a_b \in A_b$ as busy
4   $max\_gain \leftarrow -\infty$
5   $progressing \leftarrow true$
6   while progressing and not stop_condition do
7     $progressing \leftarrow false$
8     for each task $t_c \in T_c$
9       release agent $a_b$ performing task $t_c$
10      $Z_c \leftarrow Z_c \setminus \{(t_c, a_b)\}$
11      for each agent $a_f \in A$ which is free
12        assign $t_c$ to $a_f$ and set $a_f$ as busy
13        $Z_{tmp} \leftarrow Z_c \cup \{(t_c, a_f)\}$
14        $V' \leftarrow$ profit_matrix$(A, T, BPM, T_c, Z_{tmp})$
15        $Z_{nc} \leftarrow$ hungarian $(V')$
16        $gain \leftarrow u(Z_{tmp} \cup Z_{nc})$
17        if $gain > max\_gain$
18          $max\_gain \leftarrow gain$
19          $z \leftarrow (t_c, a_f)$
20          $Z \leftarrow Z_{tmp} \cup Z_{nc}$
21          $progressing \leftarrow true$
22        end if
23      end for
24      $Z_c \leftarrow Z_c \cup \{z\}$
25    end for
26  end while
```

partitioning the tasks into two sets of critical and non-critical tasks. The optimal solution for non-critical tasks is found by the Hungarian algorithm, while a heuristic optimization technique is employed to find a near-optimal solution for critical tasks. The heuristic method used in the algorithm is quite similar to *hill climbing*. Initially, critical tasks T_c are assigned to a random feasible set of agents A_b (line 3). This solution is improved in the main while-loop of the algorithm (lines 6–26). The loop is stopped if the result does not improve anymore or if it has iterated a sufficient number of times and the *stop_condition* has been set to true. In each iteration of the outer for-loop (lines 8–25), an assignment of a critical task to an agent is removed and in the inner for-loop (lines 11–23) this task is assigned to each of the non-busy agents in due order. The tasks so far assigned are called Z_{tmp} (line 13) and matrix *V'* for the rest of tasks and remaining agents is determined by calling the *profit_matrix* function (line

Table 1. Run time of the simulation in seconds – type I

Size	10^3	10^4	10^5	10^6	10^7	10^8
8	0.221	0.224	0.378	1.347	10.878	108.086
16	0.275	0.231	0.430	2.356	21.241	223.532
24	0.221	0.268	0.676	3.420	32.709	314.796
32	0.213	0.283	0.626	4.511	43.396	412.719
40	0.212	0.265	0.773	5.286	51.950	510.922
48	0.253	0.269	0.868	6.510	64.500	634.840
56	0.213	0.354	0.975	7.423	73.669	727.997
64	0.215	0.314	1.075	8.442	83.888	829.740
72	0.218	0.318	1.183	9.762	96.859	954.226
80	0.268	0.322	1.319	10.696	105.981	1044.931
88	0.294	0.396	1.426	11.972	118.044	1172.521
96	0.279	0.373	1.530	12.855	127.445	1267.337
104	0.259	0.361	1.631	14.011	138.151	1372.672
112	0.260	0.414	2.262	14.990	147.910	1481.136
120	0.260	0.414	2.224	16.006	158.211	1575.391
128	0.262	0.415	1.948	17.176	170.614	1700.674
136	0.260	0.464	2.090	18.324	180.230	1811.413
144	0.261	0.465	2.189	19.413	192.332	1920.387
152	0.263	0.463	2.240	20.211	200.070	1994.426
160	0.311	0.464	2.392	21.393	212.724	2122.605
168	0.310	0.514	2.548	23.911	229.362	2273.883
176	0.309	0.516	2.646	23.994	238.196	2386.780
184	0.317	0.515	2.760	24.891	248.126	2477.827
192	0.314	0.517	2.855	26.233	260.229	2597.218
200	0.314	0.519	2.954	27.069	270.979	2687.972

14). The Hungarian algorithm is used to find the optimal solution for the non-assigned tasks and so far non-busy agents (line 15). The total profit of the business process model is calculated (line 16) and the best assignment for the task is distinguished (lines 17–22), which is added to the assignments of critical tasks (line 24).

Our tests of the algorithm indicate that, regardless of the choice of the initial random assignment, the quality of the near-optimal solutions are generally very high when $T_c \leq T/4$, that is, they deviate from the optimal solution by less than 0.5% (see Section 7.3). However, having a smaller number of critical tasks improves the performance of the algorithm. This is due to the fact that the Hungarian algorithm always provides the optimal solution for non-critical tasks. Algorithm 6 can easily be modified to a *random restart hill-climbing* algorithm in order to improve the result.

6. Run-time of the simulation

Simulation of the business process model is the most time-consuming part of Algorithms 5 and 6, which is invoked by calling the *profit_matrix* function. To measure the run time of the simulation, a series of tests is performed and the results are summarized in Tables 1 and 2 for two different types of business process models. In both types the number of tasks is a multiple of 8, starting with 8 and up to 200 tasks. The workflow goes through all tasks sequentially; however, Gateways after tasks $4i+2,\ i=0,\ 1,\ \ldots,\ 49$ may change the workflow and send the token upstream. In the first type of model, which we from now on call *type I*, the token goes to task $4i+1$, as shown in Figure 2. In the second type of model (*type II*), the workflow is directed to task 1, that is, the first task of the model (see Figure 3).

In Table 1, the run times for models of size 8–200 and for 10^3–10^8replications are summarized. As expected, the run time increases linearly with the number of replications. More interesting is that the run time is linear in the number of tasks.

Models of type II are designed to demonstrate the worst-case scenario. Results for these models, which are summarized in Table 2, indicate that the run time grows exponentially in the input size. However, a closer look at the type II models shows that large-size models of

Table 2. Run time of the simulation in seconds – type II

Size	10^1	10^2	10^3	10^4	10^5
8	0.248	0.254	0.252	0.244	0.342
16	0.220	0.225	0.227	0.242	0.484
24	0.209	0.209	0.211	0.262	0.714
32	0.209	0.212	0.209	0.315	1.074
40	0.208	0.234	0.209	0.365	1.785
48	0.209	0.208	0.260	0.466	2.929
56	0.209	0.209	0.263	0.618	4.325
64	0.210	0.243	0.311	0.975	7.826
72	0.209	0.213	0.360	1.483	12.807
80	0.209	0.262	0.412	2.093	18.449
88	0.267	0.259	0.564	3.707	35.350
96	0.259	0.310	0.766	5.850	56.024
104	0.260	0.362	1.072	8.541	82.886
112	0.265	0.415	1.892	15.345	155.011
120	0.268	0.519	2.802	25.055	250.600
128	0.317	0.618	4.021	37.866	369.328
136	0.330	0.821	6.916	72.570	683.993
144	1.267	1.329	12.251	111.707	1132.955
152	0.363	1.947	16.972	168.701	1700.733
160	0.570	3.004	30.278	313.394	3122.768
168	0.617	4.781	49.974	518.378	5195.385
176	0.668	8.342	74.871	752.531	7537.265
184	1.330	13.443	136.050	1364.125	13,698.581
192	2.092	20.986	227.810	2264.690	>20,000
200	3.261	30.073	327.820	3355.807	>20,000

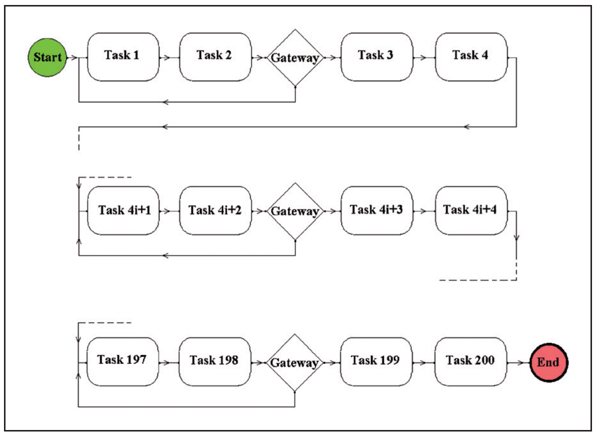


Figure 2. A type I process with 200 tasks. Gateways after task $4i+2$; $i=0, 1, \ldots, 49$ may change the workflow to task $4i+1$.

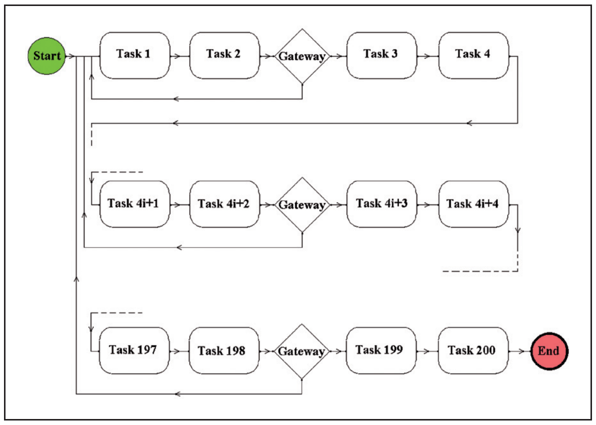


Figure 3. A type II process with 200 tasks. Gateways after task $4i+2$; $i=0, 1, \ldots, 49$ may change the workflow to task 1.

this type are not very realistic business process models. In fact, the run time of the simulation reflects the run time of the real-world business process. The increasing simulation time is due to the fact that the number of times the first tasks are performed increases exponentially. For instance, in the model with the size of 200, tasks 1 and 2 are performed 252,791.2 times on average before the process is completed.

In a real-world business process there is always a practical constraint on the number of times each task is repeated. In other words, large-size business processes of type II, for which simulation time increases exponentially in the number of tasks, are rare and can be excluded from consideration.

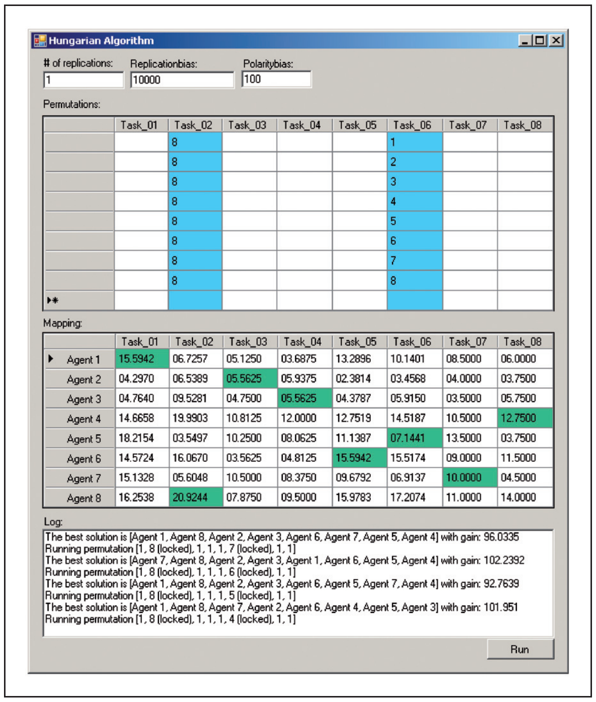
Moreover, our tests show that 10^5 simulation replications are sufficient for producing accurate results that permit the optimization algorithms to find optimal solutions.

7. Implementation and experiment results

In order to test the algorithms suggested above, we have developed an application program in *C#*, which takes advantage of models developed in the simulation software Arena.[28]

One of the features of this application is a BPMN template developed for Arena, which provides a convenient tool for modeling business processes. To run the optimization problem one should first develop the business process model in Arena and then start the application. The application reads required data from the Arena model and runs the Arena simulation when necessary. We omit the details here and refer the interested reader to Karimson.[29]

A screen shot of the application searching for the optimal solution for a business process model consisting of eight tasks is shown in Figure 4. The business process model developed in Arena is shown in Figure 5.



# of replications:	Replicationbias:	Polaritybias:
1	10000	100

Permutations:

	Task_01	Task_02	Task_03	Task_04	Task_05	Task_06	Task_07	Task_08
		8				1		
		8				2		
		8				3		
		8				4		
		8				5		
		8				6		
		8				7		
		8				8		
▸*								

Mapping:

	Task_01	Task_02	Task_03	Task_04	Task_05	Task_06	Task_07	Task_08
▸ Agent 1	15.5942	06.7257	05.1250	03.6875	13.2896	10.1401	08.5000	06.0000
Agent 2	04.2970	06.5389	05.5625	05.9375	02.3814	03.4568	04.0000	03.7500
Agent 3	04.7640	09.5281	04.7500	05.5625	04.3787	05.9150	03.5000	05.7500
Agent 4	14.6658	19.9903	10.8125	12.0000	12.7519	14.5187	10.5000	12.7500
Agent 5	18.2154	03.5497	10.2500	08.0625	11.1387	07.1441	13.5000	03.7500
Agent 6	14.5724	16.0670	03.5625	04.8125	15.5942	15.5174	09.0000	11.5000
Agent 7	15.1328	05.6048	10.5000	08.3750	09.6792	06.9137	10.0000	04.5000
Agent 8	16.2538	20.9244	07.8750	09.5000	15.9783	17.2074	11.0000	14.0000

Log:

```
The best solution is [Agent 1, Agent 8, Agent 2, Agent 3, Agent 6, Agent 7, Agent 5, Agent 4] with gain: 96.0335
Running permutation [1, 8 (locked), 1, 1, 1, 7 (locked), 1, 1]
The best solution is [Agent 7, Agent 8, Agent 2, Agent 3, Agent 1, Agent 6, Agent 5, Agent 4] with gain: 102.2392
Running permutation [1, 8 (locked), 1, 1, 1, 6 (locked), 1, 1]
The best solution is [Agent 1, Agent 8, Agent 2, Agent 3, Agent 6, Agent 5, Agent 7, Agent 4] with gain: 92.7639
Running permutation [1, 8 (locked), 1, 1, 1, 5 (locked), 1, 1]
The best solution is [Agent 1, Agent 8, Agent 7, Agent 2, Agent 6, Agent 4, Agent 5, Agent 3] with gain: 101.951
Running permutation [1, 8 (locked), 1, 1, 1, 4 (locked), 1, 1]
```

Figure 4. The application program running for eight tasks and agents.

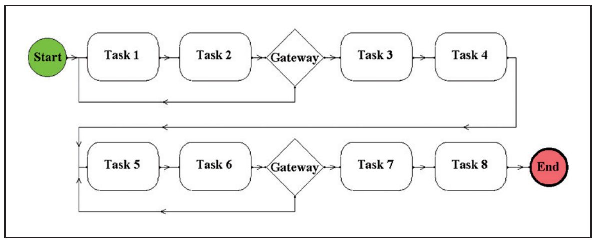


Figure 5. A process with eight tasks. Gateways after tasks 2 and 6 may change the workflow.

7.1. Test of Algorithm 3

To test the performance of Algorithm 3, a series of experiments with different problem sizes on models of type I is conducted. These models are inspired by modeling the work process of the military staff. A military staff is a group of officers in the headquarters that assist the commander in planning, coordinating and supervising operations. One of the main activities of the staff is to acquire accurate and timely information and provide the commander with analyzed and processed information. The planning process in the military staff is a complicated and lengthy process that is executed in parallel in political-military, strategic, operational and tactical levels. However, in each level the

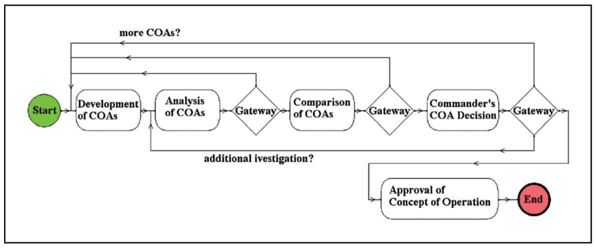


Figure 6. The concept development process within the military operational planning expressed in Business Process Modeling Notation (BPMN). COA: course of action.

process consists of several sub-processes, which are performed sequentially. Each sub-process contains a number of tasks that may be interconnected directly or through decision points. Figure 6 shows a model of the *Operational Planning Concept Development* sub-process, during which different *COAs* are evaluated.

Our test cases are built from 'blocks' of sub-processes, like the one shown in Figure 5, which has a structure similar to the aforementioned sub-process. In all these simulations, the number of tasks is a multiple of 8, starting with 8 and up to 200 tasks. The workflow goes through all tasks sequentially; however, Gateways after tasks $4i+2$, $i=0, 1,\ldots, 49$ may change the workflow and send the token upstream to task $4i+1$. The initial probability of this event is chosen randomly from 0.1, 0.2,..., 0.6. However, this probability is decreased by a value $p(n-1)/n$, where p is the initial probability and n is the number of times the token has passed tasks $4i+1$ and $4i+2$. For instance, if the probability of failure is initially 0.3, in the following attempts these probabilities will be 0.15, 0.1, 0.075,.... This assumption means that the process is not Markovian and has some kind of memory, that is, each failure to perform tasks $4i+1$ and $4i+2$ increases the probability of succeeding in future attempts.

Figures 5 and 2 show the simulation models for 8 and 200 tasks, respectively. All tasks are not shown in Figure 2, due to the lack of space. The number of agents is assumed to be equal to the number of tasks. Two experiments with two different distributions for agent capabilities are conducted. In both trails agents have four capabilities, which are graded between 0 and 5. In the first experiment these grades are drawn from a uniform distribution, while in the second experiment they are drawn from a truncated normal distribution with $\mu = 2.5$ and $\sigma = 1.0$. However, agent capabilities are always rounded to the nearest factor of 0.25.

The weight of four capabilities for tasks in both trials are drawn randomly from the values 0, 0.25, 0.5,..., 4.0 with equal probability, such that the constraint for the sum of task attributes is satisfied, that is, the sum of the attributes is equal to 4.

Agents belong to one of the three cost categories, 5, 10 or 15 (cost units per time unit), depending on whether the sum of their capabilities is less than 10.0, between 10.0 and 15.0 or greater than 15.0.

The results of Algorithm 3 for the two experiments are summarized in Tables 3 and 4. For each problem size, the gain for the optimal solution and minimum gain are presented. The number of feasible solutions for each size is also given. Execution time for the largest problem size with 200 tasks on a modest computer (Celeron 1.73 GHz processor and 2 GB RAM) is less than a few seconds.

Table 3. Results for Algorithm 3 – uniform agents

Size	Min gain	Max gain	Number of combinations
8	−152.44	442.26	4.03×10^{4}
16	−294.21	1410.52	2.09×10^{13}
24	−649.29	2608.03	6.20×10^{23}
32	−1034.58	3267.30	2.63×10^{35}
40	−1121.24	4095.89	8.16×10^{47}
48	−1363.69	5101.85	1.24×10^{61}
56	−1495.78	6125.92	7.11×10^{74}
64	−1763.97	7160.81	1.27×10^{89}
72	−2027.48	7842.16	6.12×10^{103}
80	−2331.99	8650.66	7.16×10^{118}
88	−2691.48	9710.28	1.85×10^{134}
96	−2980.34	10,554.42	9.92×10^{149}
104	−3191.85	11,555.82	1.03×10^{166}
112	−3549.29	12,481.59	1.97×10^{182}
120	−3992.73	13,623.09	6.69×10^{198}
128	−4285.26	14,654.69	3.86×10^{215}
136	−4524.24	15,648.57	3.66×10^{232}
144	−4712.97	16,634.67	5.55×10^{249}
152	−4889.44	17,544.22	1.31×10^{267}
160	−5156.85	18,417.00	4.71×10^{284}
168	−5464.02	19,689.65	2.52×10^{302}
176	−5777.10	20,571.76	1.98×10^{320}
184	−5887.74	21,548.85	2.23×10^{338}
192	−6237.12	22,526.83	3.55×10^{356}
200	−6436.01	23,542.57	7.89×10^{374}

When comparing results, one interesting observation is that the maximum gains are almost equal for both types of agents for each group size, while the minimum gain for agents chosen from a uniform distribution is significantly less than the gain for agents from a normal distribution.

7.2. Test of Algorithm 5

In order to test the performance of Algorithm 5, we use models of the type shown in Figure 2. However, we assume that tasks preceding Gateways (Task $4i+2$, $i=0, 1,\ldots$) are critical, that is, the probability of alternative paths may change depending on the performing agent. The largest problem that can be solved in a reasonable time by Algorithm 5 is one with 16 tasks. The algorithm runs $16!/12! = 43{,}680$ simulations and the same number of instances of the Hungarian algorithm, which can be compared with the number of required simulations in an exhaustive search, that is, 2.09×10^{13}. An experiment on a modest computer (Celeron

Table 4. Results for Algorithm 3 – normal agents

Size	Min gain	Max gain	Number of combinations
8	57.14	599.86	4.03×10^{4}
16	124.98	1462.34	2.09×10^{13}
24	−26.76	2470.28	6.20×10^{23}
32	−244.23	3266.29	2.63×10^{35}
40	−136.25	4044.08	8.16×10^{47}
48	−286.73	5074.53	1.24×10^{61}
56	−515.07	5863.95	7.11×10^{74}
64	−738.90	6918.70	1.27×10^{89}
72	−955.61	7817.08	6.12×10^{103}
80	−989.71	8521.75	7.16×10^{118}
88	−1265.65	9470.58	1.85×10^{134}
96	−1318.27	10,121.51	9.92×10^{149}
104	−1327.55	11,262.87	1.03×10^{166}
112	−1348.34	12,026.85	1.97×10^{182}
120	−1653.27	13,144.37	6.69×10^{198}
128	−1814.58	13,889.57	3.86×10^{215}
136	−1911.23	14,893.15	3.66×10^{232}
144	−2178.44	15,868.76	5.55×10^{249}
152	−2467.15	16,716.63	1.31×10^{267}
160	−2550.49	17,466.87	4.71×10^{284}
168	−2740.33	18,595.24	2.52×10^{302}
176	−3023.44	19,480.25	1.98×10^{320}
184	−3239.25	20,272.52	2.23×10^{338}
192	−3209.59	21,248.58	3.55×10^{356}
200	−3328.78	21,831.75	7.89×10^{374}

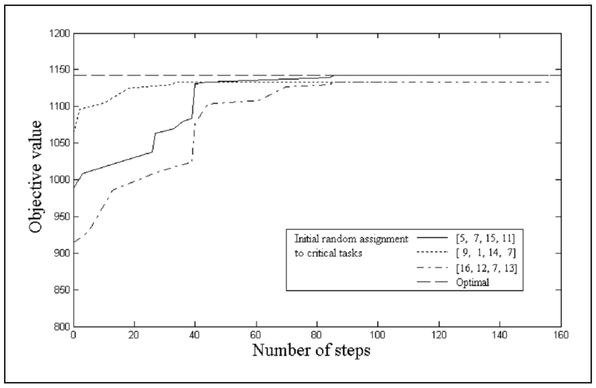


Figure 7. Result of Algorithm 6 on a model having 16 tasks showing improvement of three initial random solutions toward the optimal one.

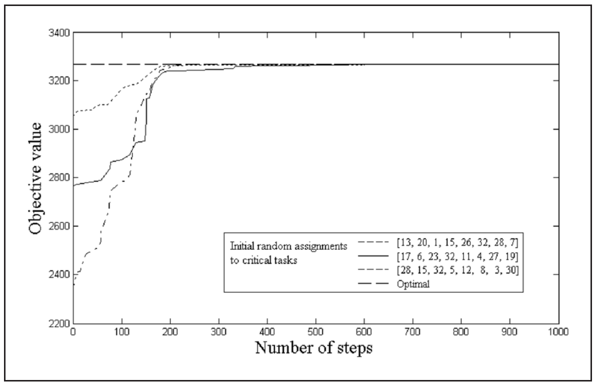


Figure 8. Result of Algorithm 6 on a model having 32 tasks showing improvement of three initial random solutions toward the optimal one.

1.73 GHz processor and 2 GB RAM), which was not especially dedicated to this task, was completed in about 2 hours. In this test each simulation was replicated 10^5 times and the average value 1141.60 for the optimal assignment was obtained.

7.3. Test of Algorithm 6

The above result (1141.60), obtained for 4 critical and 12 non-critical tasks, is compared with the results of Algorithm 6 for the same problem.

As shown in Figure 7, all three initial random solutions after a small (different) number of simulations reach a local maximum, all near the optimal value 1141.60. The rather high values of the initial random solutions are due to the fact that they are combined with the results of the Hungarian algorithm for 12 non-critical tasks. To evaluate the efficiency of Algorithm 6 on larger models, a series of tests on models with 32 and 48 tasks is conducted and the results are compared with the optimal solution. Obviously, it is not feasible to find the optimal solution by Algorithm 5 for such large models. Therefore, the models are deliberately chosen such that it is possible to find the optimal solution by Algorithm 3, that is, they have no 'real' critical tasks. These two models have the same configuration as in Figure 2, that is, they have 8 and 12 'critical' tasks, respectively.

As shown in Figures 8 and 9, the three initial random solutions evolve rather rapidly toward the value of the optimal solution. Table 5 summarizes the results of Algorithm 6 for different problem sizes, starting from one initial random solution. For each problem size, the gain of the initial random solution (start value), the best value obtained, the optimal value, the relative deviation from the optimal value and the number of steps until reaching the best value are given. For all problem sizes, the relative deviation from the optimal value is less than 0.5%.

8. Discussion

Increasing the number of critical tasks affects the performance of Algorithm 6 negatively in two ways. Firstly, a larger number of critical tasks generally results in a lower initial objective value, since assignment of a smaller portion of tasks are optimized by the Hungarian algorithm and a larger number of them are assigned purely randomly. Secondly, the final result usually has a larger gap to the optimal solution.

Figure 10 compares the results of Algorithm 6 on three business processes with 18, 36 and 54 critical tasks. All three processes have 72 tasks and the same optimal gain. The process with 18 critical tasks starts with the initial value 6501.29 and after 2468 steps reaches its best value 7808.72, and stops after 3960 steps. The process with 36 critical tasks starts with the initial value 5456.85 and after 5942 steps reaches its best value 7741.09, and stops after 7992 steps. Finally, the process with 54 critical tasks starts with the initial value 4441.40 and after 2833 steps reaches its best value 7503.37, and stops after 4104 steps. By comparing the best results of these three cases with the optimal value 7817.08, we notice that the relative deviation has increased from 0.11% to 0.97% and 4.01%, when the number of critical tasks has doubled and tripled, respectively. However, as Table 6 demonstrates, Algorithm 6 achieves high-quality solutions for different problem sizes, even when half of the tasks are critical.

It should be clarified that even though the implementation of algorithms and test results are based on models described in Equations (11) and (13), the framework and all algorithms defined in this paper are independent of these models and they can be replaced

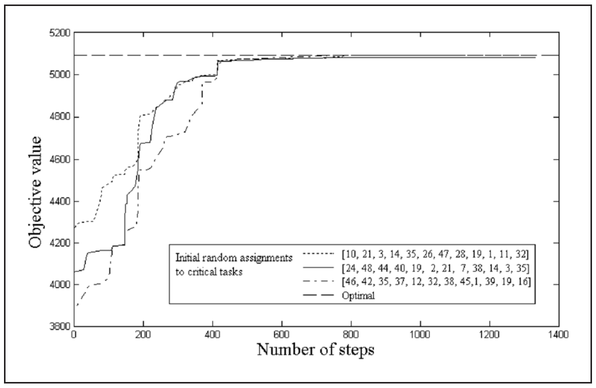


Figure 9. Result of Algorithm 6 on a model having 48 tasks showing improvement of three initial random solutions toward the optimal one.

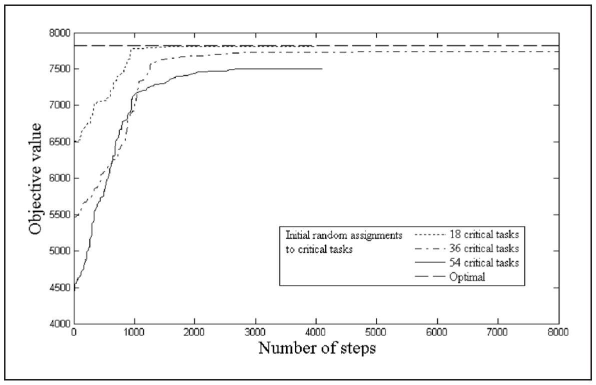


Figure 10. Comparison of the results of Algorithm 6 on three different processes with 18, 36 and 54 critical tasks.

Table 5. Results of Algorithm 6 on different problem sizes. In each model one-quarter of the tasks are critical

Size	Start value	Best value	Optimal	Relative deviation	Number of steps
8	557.79	599.86	599.86	0%	8
16	1294.88	1462.34	1462.34	0%	51
24	2162.56	2470.28	2470.28	0%	132
32	2789.84	3266.29	3266.29	0%	217
40	3553.01	4042.74	4044.08	0.03%	348
48	4396.11	5060.94	5074.53	0.27%	488
56	4999.28	5838.43	5863.95	0.44%	1062
64	5956.61	6912.29	6918.70	0.09%	2086
72	6501.29	7808.72	7817.08	0.11%	2468
80	7018.20	8515.66	8521.75	0.07%	3314
88	7691.89	9470.58	9470.58	0%	3507
96	8183.63	10,121.51	10,121.51	0%	4892
104	9182.90	11,254.43	11,262.87	0.07%	6086

Table 6. Results of Algorithm 6 on different problem sizes. In each model half of the tasks are critical

Size	Start value	Best value	Optimal	Relative deviation	Number of steps
8	500.14	599.86	599.86	0%	26
16	1261.7	1439.65	1462.34	1.56%	103
24	2073.18	2439.97	2470.28	1.23%	507
32	2605.95	3200.53	3266.29	2.01%	740
40	3324.80	3990.93	4044.08	1.31%	828
48	3962.94	4966.41	5074.53	2.13%	1934
56	4274.73	5798.11	5863.95	1.12%	2405
64	4928.07	6820.65	6918.70	1.42%	2278
72	5456.85	7741.09	7817.08	0.97%	5942
80	5957.60	8419.85	8521.75	1.20%	4474
88	6536.96	9387.58	9470.58	0.88%	5363
96	6876.37	10,039.73	10,121.51	0.81%	5740
104	7734.39	11,140.93	11,262.87	1.08%	6633

without difficulty by any other model that is validated with empirical data.

9. Conclusions and future work

In this paper we employed a model of human agents' performance in a business process to estimate an overall measure for performance of a business process model. This model is based on the agents' capabilities and the weight (importance) of these capabilities for each task. This performance metric is used to find the optimal assignment of tasks to agents. Two main categories of processes, *assignment-independent* and *assignment-dependent*, are distinguished. Each of these two categories is divided into three types, *deterministic*, *Markovian* and *non-Markovian* processes, leading to a total of six types of processes.

1. Assignment-independent deterministic processes with a predetermined workflow. The optimal solution for this type of process is found by using the Hungarian algorithm in polynomial time (Algorithm 1).
2. Assignment-independent Markovian processes. We presented an analytical method to estimate the number of times each task is performed and reduced the problem to type one, which can be solved using the Hungarian algorithm (Algorithm 2).
3. Assignment-independent non-Markovian processes, for which we used a simulation method to estimate the expected number of times each task is performed. These values are used to find the optimal solution (Algorithm 3).
4. Assignment-dependent deterministic processes.
5. Assignment-dependent Markovian processes.
6. Assignment-dependent non-Markovian processes.

In the latter three cases, a process may contain critical tasks. Critical tasks are those tasks that may affect the workflow. We introduced two algorithms for these types of processes. The first one (Algorithm 5) finds the optimal solution, but is computationally feasible only when the number of critical tasks is small. The second algorithm that is applicable to a large number of critical tasks provides a near-optimal solution (Algorithm 6). In this case, a hill-climbing heuristic method is combined with the Hungarian algorithm and simulation to find an overall near-optimal solution. The Hungarian algorithm always finds the optimal assignment for non-critical tasks and the heuristic method efficiently explores the search space to achieve a near-optimal solution. Both these methods employ simulation in order to deal with the uncertainty in the system, when the process is non-deterministic.

A series of tests that demonstrates the feasibility of the proposed algorithms is conducted. The results confirm that the algorithms perform well for at least medium-sized business processes.

One of the shortcomings of the method introduced in this paper is the model of performance of human agents, which assumes that agents assigned to a task are independent and thus it does not take into account different aspects of team working and interaction between agents. Incorporating various aspects of team working in the model is our next step in this work, which we have already initiated.

Acknowledgments

The authors would like to thank all members of the project group, especially Johan Schubert for his comments, suggestions and constructive critiques.

This paper is an extended version of the conference paper[30] presented at PADS-2010, Atlanta, GA, which was nominated for the best paper award.

Funding

This work was supported by the Totalförsvarets Forskningsinstitut (FOI) research project 'Real-Time Simulation Supporting Effects-Based Planning', which is funded by the research and development (R&D) program of the Swedish Armed Forces.

Conflict of interest statement

None declared.

References

1. Pentico DW. Assignment problems: A golden anniversary survey. *Eur J Oper Res* 2007; 176: 774–793.
2. Fisher ML, Jaikumar R and Wassenhove LNV. A multiplier adjustment method for the generalized assignment problem. *Manage Sci* 1986; 32: 1095–1103.
3. Blum C and Roli A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput Surv* 2003; 35: 268–308.
4. Goldberg DE. *Genetic algorithms in search, optimization and machine learning, Reading,* MA: Addison Wesley, 1989.
5. Mitchell M. *Introduction to genetic algorithms*. Cambridge, MA: MIT Press, 1999.
6. Holland JH. *Adaptation in natural and artificial systems*. Cambridge, MA: MIT Press, 1992.
7. Chu PC and Beasley JE. A genetic algorithm for the generalised assignment problem. *Comput Oper Res* 1997; 24: 17–23.
8. Baker BM and Ayechew MA. A genetic algorithm for the vehicle routing problem. *Comput Oper Res* 2003; 30: 787–800.
9. Gonçalves JF, Mendes JJM and Resende MGC. A genetic algorithm for the resource constrained multi-project scheduling problem. *Eur J Oper Res* 2008; 189: 1171–1190.
10. Etiler O, Toklu B, Atak M and Wilson J. A genetic algorithm for flow shop scheduling problems. *J Oper Res Soc* 2004; 55: 830–835.
11. Kirkpatrick S, Gelatt CD and Vecchi MP. Optimization by simulated annealing. *Science* 1983; 220: 671–680.
12. Cerny V. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *J Optim Theor Appl* 1985; 45: 41–51.
13. van der Aalst W, Hofstede AHMT and Weske M. Business process management: A surveyIn *Proceedings of the 1st International Conference on Business Process Management (BPM 2003).* , ser. LNCS, vol. 2678. Springer, 2003, pp.1–12.
14. Sarshar K, Theling T, Loos P and Jerrentrup M. Integrating process and organization models of collaborations through object Petri nets. In *Proceedings of Multikonferenz Wirtschaftsinformatik (MKWI 2006)*, Passau, Germany. 2006, p.329–343.
15. Object Management Group (OMG). 'Business Process Modeling Notation (BPMN) Version 1.2'http://www.bpmn.org (2009, accessed 11 January 2010).
16. Takemura T. Formal semantics and verification of BPMN transaction and compensation. In *Proceedings of the IEEE Asia-Pacific Services Computing Conference (APSCC 2006)*, Los Alamitos, CA, IEEE Computer Society. 2008, p.284–290.
17. Nicolae O, Cosulschi M, Giurca A and Wagner G. Towards a BPMN semantics using UML models. In *Proceedings of the Business Process Management Workshops*. 2008, p.585–596.
18. Smirnov S. Structural aspects of business process diagram abstraction. In *Proceedings of the IEEE Conference on Commerce and Enterprise Computing (CEC'09)*, Washington, DC, IEEE Computer Society. 2009, p.375–382.
19. Wohed P, van der Aalst WMP, Dumas M, ter Hofstede AHM and Russell N. On the suitability of BPMN for business process modelling. In: Dustdar S, Fiadeiro JL and Sheth AP (eds) *Business process management*, ser. LNCS, vol. 4102. Springer, Heidelberg, 2006, pp.161–176.
20. Siegeris J and Grasl O. Model driven business transformation – an experience reportIn *Proceedings of the 6th International Conference on Business Process Management (BPM'08).* , ser. LNCS, vol. 5240, Berlin, Heidelberg: Springer, 2008, pp.36–50.
21. Magnani M and Montesi D. *Computing the cost of BPMN diagrams*, May 2007.
22. Kamrani F, Ayani R, Moradi F and Holm G. Estimating performance of a business process model. In: Rossetti MD, Hill RR, Johansson B, Dunkin A and Ingalls RG (eds) In *Proceedings of the Winter Simulation Conference (WSC'09)*, December 2009, Austin, TX. .
23. Kuhn HW. The Hungarian method for the assignment problem. *Nav Res Logist Q* 1955; 2: 83–97.
24. Frank A. On Kuhn's Hungarian method – a tribute from Hungary. *Nav Res Logist* 2005; 52: 2–5.
25. Munkres J. Algorithms for the assignment and transportation problems. *J Soc Ind Appl Math* 1957; 5: 32–38.
26. Sasieni M, Yaspan A and Friedman L. *Operations research – methods and problems*. New York: John Wiley, 1959.
27. Jonker R and Volgenant T. Improving the Hungarian assignment algorithm. *Oper Res Lett* 1986; 5: 171–175.
28. Kelton WD, Sadowski RP and Swets NB. *Simulation with Arena*, 5th ed. New York: McGraw-Hill, 2010.
29. Karimson A. *Optimizing business processes with Arena simulation*. Master's thesis. Stockholm, Sweden: Royal Institute of Technology (KTH), December 2009.
30. Kamrani F, Ayani R and Karimson A. Optimizing a business process model by using simulationIn *Proceedings of the IEEE Workshop on Principles of Advanced and Distributed Simulation (PADS 2010)*, May 2010, Atlanta, GA. p.40–47.

Farzad Kamrani is a PhD student in the School of Information and Communication Technology at the KTH Royal Institute of Technology. He holds an MSc in Computer Science from the University of Gothenburg. His research interests are discrete event simulation, business process modeling, agent technology and simulation optimization techniques.

Rassul Ayani is professor of computer science in the School of Information and Communication Technology at the KTH Royal Institute of Technology. He received his first degree from the University of Technology in Vienna (Austria), his MSc from the University of Stockholm and his PhD from the KTH Royal Institute of Technology in Stockholm. He has been conducting research on distributed systems, distributed simulation and wireless networks since 1985. He has served as program chair and program committee member at numerous international conferences and has been an editor of the Association for Computing Machinery (ACM) Transactions on Modeling and Computer Simulation (TOMACS) since 1991.

Farshad Moradi is a senior scientist and program manager in the area of modeling and simulation, and head of the Modelling and Simulation Centre at the Swedish Defence Research Agency (FOI). He has been working on modeling and simulation for more than 16 years and has led numerous projects in this field. He holds an MSc in Computer Science and Engineering from Chalmers University of Technology, Gothenburg, Sweden, and a PhD in Distributed Simulations from the KTH Royal Institute of Technology. His research interests are in the areas of distributed systems, distributed and web-based modeling and simulation, component-based modeling and simulation and embedded simulations systems.