Performance evaluation of scheduling policies for the DRCMPSP

Ugur Satic[1 [0000−0002−9160−0006]], Peter Jacko[1 [0000−0003−3376−0260]], and Christopher Kirkbride[1 [0000−0002−3667−3413]]

Lancaster University, Lancaster LA1 4YW, UK
u.satic@lancaster@ac.uk

Abstract. In this study, we consider the dynamic resource-constrained multi-project scheduling problem where projects generate rewards at their completion, completions later than a due date cause tardiness costs and new projects arrive randomly during the ongoing project execution which disturbs the existing project scheduling plan. We model this problem as a discrete Markov decision process and explore the computational limitations of solving the problem by dynamic programming. We run and compare four different solution approaches on small size problems. These solution approaches are: a dynamic programming algorithm to determine a policy that maximises the average profit per unit time net of charges for late project completion, a genetic algorithm which generates a schedule to maximise the total reward of ongoing project and updates the schedule with each new project arrival, a rule-based algorithm which prioritise processing of tasks with the highest processing durations, and a worst decision algorithm to seek a non-idling policy to minimise the average profit per unit time. Average profits per unit time of generated policies of the solution algorithms are evaluated and compared. The performance of the genetic algorithm is the closest to the optimal policies of the dynamic programming algorithm, but its results are notably suboptimal, up to 67.2%. Alternative scheduling algorithms are close to optimal with low project arrival probability but quickly deteriorate their performance as the probability increases.

Keywords: Dynamic programming · Resource constraint · Project scheduling · DRCMPSP.

1 Introduction

Project management is crucial for many sectors such as engineering services, software development, IT services, construction and R&D, [8,5,25,1]. However it is a very challenging enterprise in that only 40% of projects are completed within time, 46% of projects are completed within their predicted budget and only 36% of projects realise their full benefit [3]. Many uncertain factors may affect project execution such as new project arrivals. In this environment, problem size grows and becomes intractable for an exact solution; thus, approximation algorithms are generally preferred. This study applies an exact solution method and some

approximation methods to project scheduling problems under uncertainty and compare their performances.

"A *project* is a unique, transient endeavour, undertaken to achieve planned objectives, which could be defined in terms of outputs, outcomes or benefits." [2]. A project consists of a collection of *tasks* that are connected via network relationships. A *reward* is released as the outcome of a project completion. A project is completed when all of its tasks are processed and an amount of *resources* (e.g. manpower, equipment) is spent over time to process these tasks. Completion of the project beyond a pre-determined *due date* or to lower standards than agreed may cause penalties and loss of prestige and goodwill which are collectively called the *tardiness cost*.

Determining a task processing order to achieve project goals such as completion in the minimum time or completion within a specific time is called *project scheduling problem* (PSP). The PSP is a vast research area which aims at optimising of project duration, resource allocation and cost evaluation [16]. In this area, the *resource-constrained project scheduling problem* (RCPSP) is one of the most extensively studied research [6]. The common goal of the RCPSP is minimising the completion time and *genetic algorithm* (GA) is the most used solution algorithm for this deterministic problem in the literature [11]. Well-known RCPSP test problems are available at PSPLIB [13], which is an online RCPSP library (http://www.om-db.wi.tum.de/psplib).

Companies usually manage multiple projects simultaneously, and the RCPSP with multiple projects is called *resource-constrained multi-project scheduling problem* (RCMPSP) [1]. The RCMPSP is a generalisation of the RCPSP, which is an NP-hard class optimisation problem; thus, RCMPSP and the other generalisation of RCPSPs are also categorised as NP-hard. [7]. Two RCMPSP solution approaches exist: (1) the first approach combines projects in parallel with a dummy start-task and a dummy end-task, then solves the problem as a giant RCPSP, (2) the second approach maintains the multiple projects separately [4]. The general goal of the RCMPSP is minimising the total (for the first approach) or the average (for the second approach) completion time [4]. A RCMPSP library named MPSPLIB is available at "http://www.mpsplib.com/" which contains sets of problems generated by Homberger [10].

The RCPSP and RCMPSP are static, where the data of project arrival times and their type are known before the scheduling begins. However, many companies accept new projects during the processing of ongoing projects [9]. That deviates from the project plan and leads to missed due dates and associated tardiness costs [5]. So, instead of focusing only on completion times, projects are modelled with completion rewards and the objective becomes to maximise the expected profit which is the difference between expected completion rewards and expected tardiness costs. The RCMPSP with uncertain project arrivals and deterministic task durations is called *dynamic* RCMPSP (DRCMPSP). Two main approaches are available for the DRCMPSP; (1) reactive baseline scheduling (e.g. [17]), an approach which generates a baseline schedule and updates it at each project

arrival which allows usage of the static RCMPSP methods such as the GA for the DRCMPSP and (2) computation of optimal policies (e.g. [18]).

In this paper, we consider the DRCMPSP with uncertain project arrivals. We model the problem as an infinite-horizon discrete-time *Markov decision process* (MDP) which is defined by five elements: time horizon, decision state space, action set, transition function and profit function. We generated task processing policies for the DRCMPSP using multiple solution methods. First, we used *dynamic programming value iteration* method to maximise the time-average profit. Second, we used GA to maximise the total completion reward and reactively fixed the schedule distribution for each project arrivals. Third, we used a *rule-based algorithm* (RBA) to generate a policy using the longest task first rule. Finally, we used *worst decision algorithm* (WDA) to generate a non-idling policy which aims to minimise the time-average profit.

We contribute to the literature by (i) developing a DRCMPSP model considering multi-task project types, extending the work of Melchiors et al. [14] who only considered single-task projects, (ii) developing an efficient implementation of the value iteration algorithm in Julia programming language to solve our model with up to 4 project types, (iii) comparing the (exactly) optimal policy of value iteration with the above-mentioned benchmark policies to evaluate the performance gap between solution approaches, and (iv) illustrating that even in simple problems with 2 or 3 project types, the suboptimality gap of benchmark policies commonly used in practice (genetic algorithm and longest-task-first rule) which ignore possibility of new project arrivals is remarkable.

This paper is organized as follows: In section 2, we describe the problem setting, the MDP model. In section 3, we describe the compared algorithms and discuss comparison results in section 4. In section 5, conclusion is presented.

2 Methodology

2.1 The problem setting

The DRCMPSP comprises J project types, and the system capacity for each project type is limited to one. All projects of type j share the same characteristics such as arrival probability (λ_j), number of tasks (I_j), task durations ($t_{j,i}$), project network, resource usages ($b_{j,i}$), project due date (F_j), reward (r_j) and tardiness cost (w_j).

A project may arrive to the system at any point during the time unit, which is the duration between two decision epochs. Only one project for each type may arrive per unit time with probability λ_j for a project of type j. Projects are stored in the system until the end of unit time. Then in the next decision epoch, if the system capacity for newly arrived project type is not full, it will get accepted to the system. Otherwise, it will get rejected.

A type j project consists of I_j tasks. In this problem, tasks are connected sequentially with a successor-predecessor relationship, which defines the project network. Processing task i of project type j requires completion of its predecessor

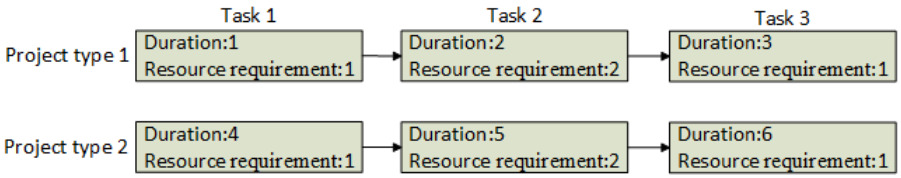

Fig. 1. A project network

tasks ($\mathcal{M}_{j,i}$) which have an earlier place in the project network. An example project network is shown in Figure 1.

Processing task i from project type j also requires allocation of $b_{j,i}$ amount of resources during its processing. Only one type of resource is defined in our model and the amount available is represented by B. The total number of allocated resources cannot be higher than B. The resources are assumed renewable which means they become reusable after completion of a task to which they were assigned. The number of resources which are not allocated for task processing is called free-resources (B_s^{free}). After the completion of a task, its allocated resources return to the free resources.

Task processing is assumed to be non-preemptive; thus, it cannot be paused or cancelled. i.e., once a task has begun processing, it does not leave processing until completed.

Projects are completed when all of their tasks are processed, and a project reward r_j is earned. Projects have a due date (F_j) which represents the maximum unit of time which can be spent for project completion to obtain its full reward r_j. If the due date is exceeded, the tardiness cost w_j is applied only once, after the project is completed.

2.2 Decision State

The decision state (s) represents the system information relevant to the decision-making process at each decision epoch [21]. Decision states where the resource limitations are not exceeded and predecessor tasks were completed before their successor tasks are called feasible and the set of all feasible decision states is called as the state space $\mathcal{S}$. Elements of a decision state ($s = \{\boldsymbol{P}_1, \boldsymbol{P}_2 \ldots, \boldsymbol{P}_J\}$) are project states ($\boldsymbol{P}_j$) for all project types. A project state consists of task states ($x_{j,i}$) and the remaining due date state (d_j) ($\boldsymbol{P}_j = (x_{j,1}, x_{j,2}, \ldots, x_{j,I_j}, d_j)$).

A task state ($x_{j,i} \in \{-1, 0, 1, 2, \ldots, t_{j,i} - 1\}$) represents the status of a task. If a task is pending for processing, its value is taken as -1. If a task is finished, its value is represented by 0. If a task is in processing, its value is the remaining processing time to its completion.

The remaining due date state ($d_j \in \{0, 1, 2, 3, \ldots, F_j\}$) represents the number of remaining time units from the current time epoch to complete the project j without paying any tardiness cost. When a due date is exceeded, its value becomes 0 and it expresses that the tardiness cost will be incurred at the project's

Table 1. State Matrix

		Remaining duration of task i			Remaining due date
		1	2	3	d
Project j	1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	d_1
	2	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	d_2

completion. A newly accepted project has the highest remaining due date state value which is its due date F_j.

When a type j project is completed or there is no type j project in the system ($\boldsymbol{P}_j = (0, 0, \ldots, 0, 0)$), all task states ($x_{j,i} = 0, \forall i$) and remaining due date state ($d_j = 0$) of project type j are represented by 0.

When a new type j project arrives ($\boldsymbol{P}_j = (-1, -1, \ldots, -1, F_j)$), all its task states are set to -1 ($x_{j,i} = -1, \forall i$) and its remaining due date state is represented by its due date F_j ($d_j = F_j$).

An example state matrix with two projects and three tasks is shown in Table 1. Here, rows of the matrix represent each project type j. The columns represent the task numbers but the last column of the matrix represents the due date state (d_j).

A decision state determines its free resources (B_s^{free}) which is used as a constraint for the available decisions. Free resources are the remaining resources available after the resource allocation to ongoing tasks has been accounted for:

$$B_s^{\text{free}} = B - \sum_{j=1}^{J} \sum_{i=1}^{I_j} b_{i,j} \mathcal{I}\{x_{i,j} > 0\} \tag{1}$$

Here, B is the total amount of resource, $b_{i,j}$ is the resource amount allocated for processing of task i from type j project, $\mathcal{I}\{.\}$ is an indicator function that takes the value 1 if the condition in parentheses is true and takes the value 0 otherwise.

2.3 Action Representation

The decisions available in a given decision state s is called an action a. At a decision epoch, the decision maker selects an action a which starts the processing of the selected pending tasks. An example action matrix with two projects and three tasks is shown in Table 2. If the decision includes processing a pending task i of a type j project ($x_{i,j} = -1$), the corresponding action element $a_{j,i}$ will take the value of 1 in the action matrix. Otherwise, $a_{j,i}$ will be 0. The task processing decision can be only taken if there are enough free resources to allocate ($\sum_{j=1}^{J} \sum_{i=1}^{I_j} b_{i,j} \mathcal{I}\{a_{i,j} = 1\} \leq B_s^{\text{free}}$) and any predecessor tasks ($\mathcal{M}_{j,i}$) of task i are completed ($\sum_{m \in \mathcal{M}_{j,i}} x_{j,m} = 0$). Thus, an action must satisfy both of these conditions. All the actions which meet both the resource and predecessor

Table 2. Action Matrix

		Task i		
		1	2	3
Project j	1	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
	2	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$

limitations, are called feasible and set of all feasible actions for a decision state s creates the action set ($\boldsymbol{A}(s) = \{\mathbf{0}, \boldsymbol{a}^{'}, \boldsymbol{a}^{''}, \dots\}$).

The action, where all action elements are zero "do not initiate any task" ($\mathbf{0} = (0, 0, \dots, 0)$) and it is always a member of the action set $\mathbf{0} \in \boldsymbol{A}(s)$. $\boldsymbol{a}^{'}$ and $\boldsymbol{a}^{''}$ represent alternative feasible actions. The number of alternative actions in an action set depends on the number of free resources (B_s^{free}), the unprocessed tasks (with $x_{i,j} = -1$) and the tasks with completed predecessor tasks (with $\sum_{m \in \mathcal{M}_{j,i}} x_{j,m} = 0$).

2.4 Transition function

The transition function describes how the system evolves from one state to another as a result of decisions and information [19]. The period between two consecutive decision states is the time unit. During the transition period; the ongoing tasks are processed for one time unit, some tasks are completed and new projects may arrive according to arrival probabilities λ_j. The project arrival probability is considered when a project is to be completed before the next decision epoch (i.e., the system capacity for type-j project will become available). The transition function is defined in Equation 2.

$$P(s'|s,a) = \prod_{j=1}^{J} \prod_{i=1}^{I_j} P(x'_{j,i}|x_{j,i}) \tag{2}$$

$$P(x'_{j,i}|x_{j,i}) = \begin{cases} \lambda_j, & \text{for } 0 \le x_{j,i} \le 1, x'_{j,i} = -1, i = I_j \\ \lambda_j, & \text{for } x_{j,i} = -1, a_{j,i} = 1, x'_{j,i} = -1, i = I_j \\ 1 - \lambda_j, & \text{for } 0 \le x_{j,i} \le 1, x'_{j,i} = 0, i = I_j \\ 1 - \lambda_j, & \text{for } x_{j,i} = -1, a_{j,i} = 1, x'_{j,i} = 0, i = I_j \\ 1, & \text{for } x_{j,i} \ge 2, x'_{j,i} = x_{j,i} - 1, i = I_j \\ 1, & \text{for } x_{j,i} = -1, a_{j,i} = 0, x'_{j,i} = -1, i = I_j \\ 1, & \text{for } x_{j,i} \ge -1, x'_{j,i} \ge -1, i < I_j \end{cases}$$

In Figure 2 an example transition process has been shown. The transition probability of the first alternative future decision state, where the last task of type j project is finished and a new type j project arrived, is $P(s''|s,a) = \lambda_j$. The transition probability of the second alternative future decision state, where the last task type j project is finished and no project arrived, is $P(s'|s,a) = (1-\lambda_j)$.

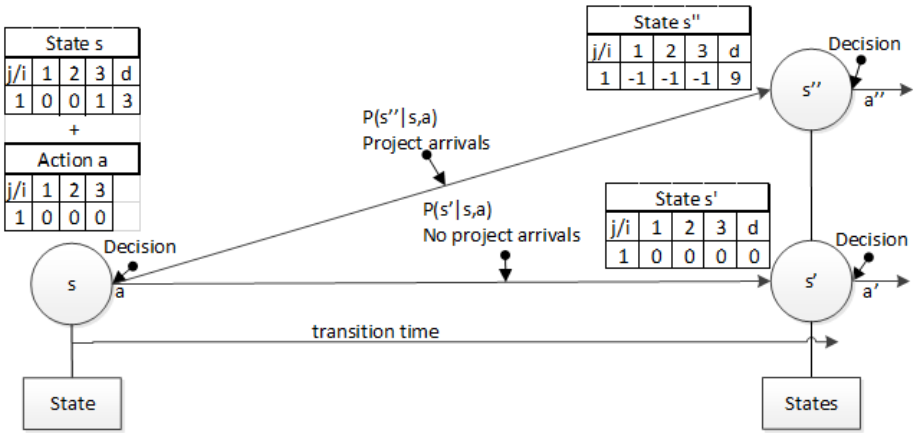

Fig. 2. A state transition diagram (for a $j = 1$ type project with 3 tasks ($i = 1, 2, 3$) whose due date is $F_j = 9$ and the selected action means do not initialise any task.)

2.5 Profit Representation

The profit function ($R_{s,a}$) is the sum of rewards (r_j) of completed projects in the period between current and next decision epoch minus the tardiness cost of late completions which depend on the remaining due dates.

$$R_{s,a} = \sum_{j=1}^{J} r_j \mathbb{E}\Big[\mathcal{I}\{x_{j,I} = 1 \vee (x_{j,I} = -1 \wedge a_{j,I} = 1 \wedge t_{j,I} = 1)\}\Big] - \sum_{j=1}^{J} w_j \mathbb{E}\Big[\mathcal{I}\{x_{j,I} = 1 \vee (x_{j,I} = -1 \wedge a_{j,I} = 1 \wedge t_{j,I} = 1) \wedge d_j = 0\}\Big] \tag{3}$$

Here, the first indicator is for project completion and takes the value 1 if a project completes and is 0 otherwise. The second indicator is for late project completion. It takes the value 1 if a project's due date has already passed (i.e., the projects remaining dues date $d_j = 0$) and is 0 otherwise. Recall that, in decision state s, $x_{j,I}$ represents the remaining processing time of the final task of a type j project and $a_{j,I}$ its the action element under action a. $t_{j,I}$ is duration of task i of project type j.

2.6 Goal function

The goal of the DRCMPSP is to find the policy π that maximises the long-term average profit per unit time.

$$g^* = \max_{\pi \in \Pi} \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}^{\pi}[R_{s(t),a(t)}] \tag{4}$$

Here, t is the time epoch. $R_{s(t),a(t)}$ is the profit function dependent of time epoch t. π is a policy from the set of all feasible non-anticipating policies (Π) presenting the action set $\boldsymbol{A}(s)$. A feasible policy is a sequence of action which considers both the resource limitation and project network.

2.7 Solution by Dynamic Programming

Dynamic Programming is a collection of algorithms which calculates optimal policies from the MDP model of the solution environment [23]. In this research we used Dynamic Programming Value Iteration. Value Iteration calculates a sequence of value functions [24]. The value function approximates the cumulative reward minus the tardiness cost. The per-period change in the value function approximates the maximum long-term average profit. The process steps of the algorithm are below;

For each state $\forall s \in \mathcal{S}, V^{old}(s) = 0$
Do
  For each state $\forall s \in \mathcal{S}$
    $V(s) = \max_{a \in A}[R_{s,a} + \sum_{s' \in \mathcal{S}} p(s'|s,a) V^{old}(s')]$
  End For
  $W_{max} = \max_{s \in \mathcal{S}}[V(s) - V^{old}(s)]$
  $W_{min} = \min_{s \in \mathcal{S}}[V(s) - V^{old}(s)]$
  $\Delta = W_{max} - W_{min}$
  Update for $\forall s \in \mathcal{S}, V^{old}(s) = V(s)$
While $\Delta > \beta \times W_{min}$

Here, V represents the value function of a decision state s. $R_{s,a}$ is profit function as explained in subsection 2.5. $p(s'|s,a)$ is the state transition probability. s' stands for the future decision state of s. $V^{old}(s')$ is the value of s' from next decision epoch. β is pre-specified tolerance number (0.000001). W_{min} and W_{max} are respectively minimum and maximum value changes between two iterations. Δ is the difference between the minimum and the maximum value changes. $\mathcal{S}$ is the state space which is defined at subsection 2.2. These processes are repeated until the stopping criteria is met.

3 Results and Comparisons

We used two heuristic algorithms with reactive scheduling and one worst decision algorithm to compare their performance to optimal. A reactive scheduling method generates decisions within a deterministic approach without considering the future uncertainties [17]. Then, it iteratively fixes its first schedule according to random changes and makes the schedules feasible again [20]. We used a genetic algorithm and a priority rule algorithm with the reactive scheduling method.

3.1 Genetic Algorithm

The discrete-time MDP is considered as a reactive scheduling system by generating a new baseline schedule for each decision state. The baseline schedules are generated by a genetic algorithm (GA) which seeks to maximise the profit and minimising the total completion time. We adapted GA from Satic [22]. The GA is one of the search algorithms which searches for the global optimum on the solution space by improving the search samples at each iteration [15]. The GA uses bio-inspired operators (e.g. Elitist selection, Crossover and Mutation) to develop the population, which is a solution set, in each iteration.

For each decision state, random numbers are assigned to unprocessed tasks, and this assignment is stored as an individual of the population. Individuals are created until the population number (here, one hundred) is reached. The random numbers represent task processing priorities and this method called as the random key representation. The random keys are converted to a schedule using the serial scheduling scheme as Kolisch and Hartmann [12] described. Then the population is ordered according to their total profit and total completion time.

The first population is iterated one hundred times using the genetic operators. The best ten percent of the population is transferred to the next population without any change, and the rest of the next population is created with the crossover operator. The crossover operator, firstly, selects two individuals from the previous population, then, copies some random keys from the first individual, after that, copies the rest from another individual, and finally, creates a new individual. The new individual is mutated with a fifty per cent probability before joining to the next population. The mutation operator randomly selects an unprocessed task and re-assigns its random number. When the new population reaches one hundred individuals, the random keys are converted to schedules with the serial scheduling scheme and the population is ordered again according to their total profit and total completion time. After the one-hundredth generation is created; the best schedule is selected as the baseline schedule. Then the baseline schedule is converted to action.

3.2 Priority rule (Longest task first)

An alternative policy is created with a priority based heuristic algorithm. The algorithm uses a single-pass priority rule called the longest task first rule. Single-pass rules generate only one action for the given state. The rule based algorithm (RBA) prioritises the tasks with the longer processing times and if two tasks have the same duration, the smallest numbered project type, e.g., project type 1 is prioritised over type 2 or type 3. For each decision state, the algorithm generates a baseline schedule using the priority rule and the serial scheduling scheme. Then, the baseline schedule is converted to an action.

3.3 Worst decision algorithm

A mix of value iteration and priority rule methods are used as the worst decision algorithm (WDP) which seeks a policy (π') to get the minimum profit per unit time.

$$g' = \min_{\pi' \in \Pi'} \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}^{\pi'}[R_{s(t),a(t)}]. \tag{5}$$

Here, π' is a policy from the set of all feasible non-anticipating active policies (Π') which does not include the "to do not active any task" ($\mathbf{0}$) actions unless it is the only possible action in the action set ($|\boldsymbol{A}(s)| = 1$). Since the reward and tardiness costs are modelled to be received after project completions, a minimum profit algorithm without the priority rule ($|\boldsymbol{A}(s)| \neq 1 \Rightarrow \mathbf{0} \notin \pi'$) delays project completions infinitely to halt rewards.

4 Computational results

4.1 Experimental setup

In this section, we explore the limits of DP on the DRCMPSP, and compare its performance with the two heuristic reactive baseline scheduling algorithms and one worst decision algorithm. The DP and the compared algorithms are coded in JuliaPro 1.0.1.1. All tests are performed on a desktop computer with Intel i5-6500T CPU with 2.50 GHZ clock speed and 32 GB of RAM.

We generate four DRCMPSPs (see Table 3). For each project in the experiment, a project's tasks are performed in sequential numerical order, i.e., a project starts with task one which is a predecessor of task two which is a predecessor of task three. See Figure 1. The problems vary by number of projects, number of tasks, resource usage, different reward-tardiness cost settings and length of due date. We call the difference between a project's due date and the sum of tasks durations as slack time. This value also varies for each project in the problems. The total resource capacity is taken $B = 3$ for all problems.

The first problem has two project types, and each type has two tasks. Project type two has a higher completion reward and higher tardiness cost with a shorter slack time. That means while project type two contributes higher reward opportunities, its late completion is less rewarding compared to the late completion of the project type one.

The second problem has two project types, and each type has three tasks. The project type one is as twice as profitable. However, the slack time of project type one is shorter, so its due date may easily be exceeded leading to tardiness cost.

The third problem has three projects types, and each type has two tasks. In this problem, resource capacity allows parallel processing for only up to two projects increasing the chance of tardiness costs from one project. Only project type one can be processed with other types.

Table 3. Problem Parameters

2 projects and 2 tasks problem						
Project no	Reward	Tardiness cost	Due date	Task no	Task duration	Resource usage
1	3	1	8	1	2	2
				2	2	2
2	10	9	5	1	3	1
				2	1	3

2 projects and 3 tasks problem						
Project no	Reward	Tardiness cost	Due date	Task no	Task duration	Resource usage
1	12	8	10	1	1	1
				2	2	2
				3	5	1
2	6	5	15	1	4	1
				2	3	2
				3	4	1

3 projects and 2 tasks problem						
Project no	Reward	Tardiness cost	Due date	Task no	Task duration	Resource usage
1	8	5	10	1	5	1
				2	2	1
2	5	3	8	1	1	2
				2	3	1
3	20	19	10	1	2	3
				2	7	2

4 projects and 2 tasks problem						
Project no	Reward	Tardiness cost	Due date	Task no	Task duration	Resource usage
1	18	3	4	1	5	2
				2	1	1
2	27	4	5	1	4	2
				2	2	1
3	18	5	6	1	3	2
				2	3	1
4	18	6	7	1	2	2
				2	4	1

*Resource capacities = 3

The fourth problem has four projects types, and each type has two tasks. The slack times of project types one and two are negative, and project type three's slack times is zero and project type four's slack time is one. Thus most of the projects will be completed later than their planned due date, and the tardiness payment will be inevitable.

We test each problem consecutively from 1% to 90% project arrival probabilities, increment by 10%. 0% and 100% arrival probabilities are not used in this comparison, because 0% arrival probability makes the problem static and

Table 4. Comparison of the time-average profit deviations from the optimal results of DP (how much percent lower than optimal results of DP)

Project arrival probability										
	1%	10%	20%	30%	40%	50%	60%	70%	80%	90%
2 projects and 2 tasks problem										
GA	0.7%	6.5%	11.7%	15.4%	18.1%	20.0%	21.2%	21.4%	20.4%	17.0%
RBA	2.1%	19.9%	35.2%	46.1%	53.7%	59.3%	63.7%	67.3%	70.4%	72.7%
WDP	2.8%	25.6%	43.8%	55.4%	62.7%	67.3%	70.2%	72.1%	73.5%	75.5%
2 projects and 3 tasks problem										
GA	0.1%	4.9%	13.0%	22.0%	31.1%	39.1%	45.6%	51.6%	58.1%	67.2%
RBA	1.5%	15.3%	25.1%	30.1%	32.3%	32.6%	31.1%	28.2%	23.5%	15.4%
WDP	4.1%	34.3%	49.9%	59.0%	66.4%	72.6%	77.1%	80.2%	82.2%	83.3%
3 projects and 2 tasks problem										
GA	0.1%	4.9%	13.0%	22.0%	31.1%	39.1%	45.6%	51.6%	58.1%	67.2%
RBA	1.5%	15.3%	25.1%	30.1%	32.3%	32.6%	31.1%	28.2%	23.5%	15.4%
WDP	4.1%	34.3%	49.9%	59.0%	66.4%	72.6%	77.1%	80.2%	82.2%	83.3%
4 projects and 2 tasks problem										
GA	0.0%	1.2%	2.9%	5.8%	6.9%	6.8%	8.0%	11.5%	15.4%	19.0%
RBA	0.4%	6.6%	14.6%	21.4%	25.1%	26.8%	28.7%	31.4%	33.9%	36.1%
WDP	1.4%	21.3%	37.8%	46.2%	50.5%	52.8%	54.8%	57.3%	59.4%	61.5% [a]

[a] approximate

100% arrival probability causes a non-ergodic MDP, e.g., the empty state where no project has arrived cannot be reachable again from any states.

4.2 Discussion

DP suffers from "the curse of dimensionality" which means, here, the number of states grows exponentially with the number of tasks in a project, the number of project types, task durations and due dates, and the large state space becomes computationally intractable [23]. The model uses the state space as defined in subsection 2.2. In our experiment, a state space for more than five project types with two tasks each becomes computationally intractable. Thus the considered problems are limited to four projects and two tasks.

The results shown in Table 4 illustrate that the GA produces almost optimal solutions in 1% arrival rate and produces close to optimal solutions with other low arrival rates. The GA's results are generally closer to optimum compared to RBA for the majority of the considered problems and their task duration variations. The GA's results were from 0.003% to 67.2% lower than the optimum results but never exactly the same.

The RBA's results are between the GA and the WDP for most of the test problem. The RBA's results were from 0.4% to 72.7% lower than the optimum results. In three projects with two tasks problem, the RBA produced better

results than the GA at higher arrival probabilities. However, in most of the cases, its results were closer to the WDP than the optimum since the used priority rule is not designed for reward maximising.

Since the GA and the RBA are reactive baseline scheduling algorithms, they generate decisions without considering the new project arrivals. Thus we may accept that the result of a reactive baseline scheduling algorithm deteriorates compared to the optimum as problem deviates from the static assumption i.e. no project arrivals. However, some anomalies were observed for very high arrival probabilities. These anomalies occur since the tardiness cost is only paid once when a project is completed. In the current model, high arrival probabilities lead to postponing some projects infinitely. Thus, they stay in the system without causing a tardiness cost while the other projects continue processing without causing much tardiness cost.

5 Conclusion

In this paper, we studied the resource-constrained multi-project scheduling problem with uncertain project arrivals. We modelled the problem as an infinite-horizon discrete-time MDP. New project arrivals happen during the time unit. We used DP value iteration to maximise the long-term average profit per unit time. We tested the limits of the DP on the DRCMPSP and generated four test problems. We used two heuristic reactive baseline scheduling methods and a worst-decision DP on the same problems and compared their results with exact results of the DP. We used GA and RBA as heuristic reactive baseline scheduling methods.

According to our findings, GA produced closer to optimal results than the simpler heuristic RBA. Since reactive baseline scheduling does not consider the random changes before they occurred, the GA's and the RBA's results are closer to optimal at low arrival probabilities, and diverge from optimum at the high arrival probabilities.

In this work, we have seen that DP suffers from the curse of dimensionality even for the small size problems and reactive baseline scheduling methods do not produce close to optimum results at the high arrival probabilities. Therefore, as a future research topic, we suggest to use a technique which will not (or less) suffer from the curse of dimensionality but will consider the new project arrivals during the decision phase.

References

1. Adhau, S., Mittal, M.L., Mittal, A.: A multi-agent system for distributed multi-project scheduling: An auction-based negotiation approach. Engineering Applications of Artificial Intelligence **25**(8), 1738–1751 (2012). https://doi.org/10.1016/j.engappai.2011.12.003
2. APM: APM body of knowledge. Association for Project Management, 6th edn. (2012)
3. APM: The state of project management annual survey 2018 (2018), http://www.wellingtone.co.uk/wp-content/uploads/2018/05/The-State-of-Project-Management-Survey-2018-FINAL.pdf
4. Browning, T.R., Yassine, A.A.: Resource-constrained multi-project scheduling: Priority rule performance revisited. International Journal of Production Economics **126**(2), 212–228 (2010). https://doi.org/10.1016/j.ijpe.2010.03.009
5. Capa, C., Kilic, K.: Proactive project scheduling with a bi-objective genetic algorithm in an r&d department. In: 2015 International Conference on Industrial Engineering and Operations Management (IEOM). vol. 1, pp. 1–6 (2015). https://doi.org/10.1109/IEOM.2015.7093733
6. Creemers, S.: Minimizing the expected makespan of a project with stochastic activity durations under resource constraints. Journal of Scheduling **18**(3), 263–273 (2015). https://doi.org/10.1007/s1095
7. Gonçalves, J.F., Mendes, J.J., Resende, M.G.: A genetic algorithm for the resource constrained multi-project scheduling problem. European Journal of Operational Research **189**(3), 1171–1190 (2008). https://doi.org/10.1016/j.ejor.2006.06.074
8. Grey, J.R.: Buffer techniques for stochastic resource constrained project scheduling with stochastic task insertions problems. Ph.D. thesis, University of Central Florida (2007), http://purl.fcla.edu/fcla/etd/CFE0001584
9. Herbots, J., Herroelen, W., Leus, R.: Dynamic order acceptance and capacity planning on a single bottleneck resource. Naval Research Logistics (NRL) **54**(8), 874–889 (2007). https://doi.org/10.1002/nav.20259
10. Homberger, J.: A (μ, λ)-coordination mechanism for agent-based multi-project scheduling. OR spectrum **34**(1), 107–132 (2012)
11. Karam, A., Lazarova-Molnar, S.: Recent trends in solving the deterministic resource constrained project scheduling problem. In: 9th International Conference on Innovations in Information Technology (IIT). pp. 124–129 (03 2013). https://doi.org/10.1109/Innovations.2013.6544405
12. Kolisch, R., Hartmann, S.: Heuristic Algorithms for the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis, pp. 147–178. Springer US (1999). https://doi.org/10.1007/978-1-4615-5533-9_7
13. Kolisch, R., Sprecher, A.: Psplib-a project scheduling problem library: Or software-orsep operations research software exchange program. European Journal of Operational Research **96**(1), 205–216 (1997). https://doi.org/10.1016/S0377-2217(96)00170-1
14. Melchiors, P., Leus, R., Creemers, S., Kolisch, R.: Dynamic order acceptance and capacity planning in a stochastic multi-project environment with a bottleneck resource. International Journal of Production Research **56**(1-2), 459–475 (2018). https://doi.org/10.1080/00207543.2018.1431417
15. Mori, M., Tseng, C.C.: A genetic algorithm for multi-mode resource constrained project scheduling problem. European Journal of Operational Research **100**(1), 134–141 (1997). https://doi.org/10.1016/S0377-2217(96)00180-4

16. Ortiz-Pimiento, N.R., Diaz-Serna, F.J.: The project scheduling problem with non-deterministic activities duration: A literature review. Journal of Industrial Engineering and Management (JIEM) **11**(1), 116–134 (2018). https://doi.org/10.3926/jiem.2492
17. Pamay, M.B., Bülbül, K., Ulusoy, G.: Dynamic resource constrained multi-project scheduling problem with weighted earliness/tardiness costs. In: Pulat, P., Sarin, S., Uzsoy, R. (eds.) Essays in Production, Project Planning and Scheduling, vol. 200, pp. 219–247. Springer (2014). https://doi.org/10.1007/978-1-4614-9056-2_10
18. Parizi, M.S., Gocgun, Y., Ghate, A.: Approximate policy iteration for dynamic resource-constrained project scheduling. Operations Research Letters **45**(5), 442–447 (2017). https://doi.org/10.1016/j.orl.2017.06.002
19. Powell, W.B.: Approximate Dynamic Programming: Solving the Curses of Dimensionality, Wiley series in probability and statistics, vol. 842. Wiley (2011). https://doi.org/10.1002/9781118029176
20. Rostami, S., Creemers, S., Leus, R.: New strategies for stochastic resource-constrained project scheduling. Journal of Scheduling **21**(3), 349–365 (2018). https://doi.org/10.1007/s1095
21. Sammut, C., Webb, G.I. (eds.): Decision Epoch. Springer US (2010). https://doi.org/10.1007/978-0-387-30164-8_198
22. Satıç, U.: Çok kaynak kısıtlı projelerin sezgisel yöntemlerle çizelgelenmesi. Yildiz Technical University (2014), https://tez.yok.gov.tr/UlusalTezMerkezi/TezGoster?key=gyLHMouPes-CvnhRcjQsKQIo1AYi4WhfoZxbAaK4INn4yvJXWUop3HXf7wZV_sh4
23. Sutton, R.S., Barto, A.G., Bach, F.: Reinforcement Learning: An Introduction. Adaptive Computation and Machine Learning, MIT Press, second edition. edn. (2018)
24. Tijms, H.C.: Stochastic models: an algorithmic approach. John Wiley & Sons (1994)
25. Wang, X., Chen, Q., Mao, N., Chen, X., Li, Z.: Proactive approach for stochastic rcmpsp based on multi-priority rule combinations. International Journal of Production Research **53**(4), 1098–1110 (2015). https://doi.org/10.1080/00207543.2014.946570