Nash equilibrium solutions in multi-agent project scheduling with milestones

Přemysl Šůcha, Alessandro Agnetis, Marko Šidlovský, Cyril Briand

Version of Record: https://www.sciencedirect.com/science/article/pii/S0377221721000400
Manuscript_87de7d383a2300d9f54b857c1a862165

Nash equilibrium solutions in multi-agent project scheduling with milestones

Přemysl Šůcha[a,c,*], Alessandro Agnetis[b], Marko Šidlovský[a], Cyril Briand[c]

[a] *Czech Technical University in Prague*
Czech Institute of Informatics, Robotics, and Cybernetics
Jugoslávských partyzánů 1580/3, 160 00, Prague, Czech Republic

[b] *Università degli Studi di Siena*
Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche
via Roma 56, 53100 Siena, Italy

[c] *LAAS-CNRS, Université de Toulouse, CNRS, UPS*
7 Avenue du Colonel Roche, 31400 Toulouse, France

Abstract

This paper addresses a project scheduling environment in which the activities are partitioned among a set of agents. The project owner is interested in completing the project as soon as possible. Therefore, she/he defines rewards and penalties to stimulate the agents to complete the project faster. The project owner offers a per-day reward for early project completion and defines intermediate project milestones to be met within specific due dates, with associated per-day penalties. Each agent can, therefore, decide the duration of her/his activities, taking into account linear activity crashing costs, the reward for early project completion, and the penalty arising from violating milestone due-dates. We consider Nash equilibria, i.e., situations in which no agent has an interest in individually changing the duration of any of her/his activities, and in particular, the problem of finding a minimum-makespan equilibrium. This problem is known to be NP-hard, and in this paper, we (*i*) propose a new and efficient exact algorithmic approach for finding the minimum-makespan equilibrium and (*ii*) through an extensive computational campaign we evaluate the role played by milestones in driving the project towards the owner's goal.

Keywords: project scheduling, Nash equilibria, flow networks, milestones, lazy-constraint generation.

*Corresponding author
Email addresses: suchap@cvut.cz (Přemysl Šůcha), agnetis@diism.unisi.it (Alessandro Agnetis), marko.sidlovsky@gmail.com (Marko Šidlovský), briand@laas.fr (Cyril Briand)

1. Introduction

This paper addresses a multi-agent project scheduling problem having many stakeholders. The stakeholders are the *project owner* (or contractor, investor) managing the project and several *agents* (such as subcontractors, firms, organizations) that participate in carrying out a large number of activities of a project. This situation is typical of large-size collaborative projects in civil engineering, building industry, aeronautical or naval construction. In such project settings, key factors to project success include synchronization among the various stakeholders, correct allocation of resources and careful progress monitoring, which can be facilitated by appropriate milestone setting (e.g. [Spalek 2005] includes project monitoring and resource planning among factors having large impact on project success). This work explicitly acknowledges the presence of stakeholders, intermediate project milestones, and the financial penalties each agent has to bear if some milestone is reached late in the project.

The *project* consists of a set of *activities*, interconnected by precedence relations, designed to reach a particular goal. The activities are carried out by the *agents*, each having exclusive control over a subset of activities. The assignment of activities to agents is given, as it reflects the various agents' skills and competencies. A *normal* duration is specified for each activity. Paying a certain per-day *crashing cost*, the owner of the activity may decide to shorten it (possibly up to a minimum *crash duration*). When all activities have normal duration, the project owner attains a certain *normal makespan*. In this paper, we adopt the project owner's point of view. The project owner is interested in completing the project as soon as possible. Therefore he/she offers each agent a fixed *reward* (in general different for individual agents) for each day of makespan decrease with respect to the normal makespan. This paper introduces further leverage through which the project owner can achieve control over project implementation. We consider that some nodes of the project network are specified as *milestones*, and each of them has a specific *due date*. If a milestone is reached after its due date, each agent pays a certain per-day *penalty*, while no penalty is incurred if the due date is not violated. Hence, the total *profit* of an agent results from the balance between the reward attained from early project completion, the payments related to activity crashing, and the penalties associated with milestones' due date violations.

We model a project scheduling problem related to a classical noncooperative game setting, in which each agent's strategy consists of deciding the duration of his/her activities. As in classical noncooperative games, our analysis aims at finding Nash equilibria, i.e., situations in which no agent has an interest in changing her/his activities' durations if no other agent does so. To this aim, complete knowledge of the game parameters is generally assumed. The primary motivation for this approach is that, by computing a minimum-makespan Nash equilibrium, the project owner (who is interested in complying with the milestones and minimizing project completion time) is able to prescribe (or negotiate) *stable* activity durations. For the purpose of our analysis, it is worth mentioning that an agent does not need to know the rewards or crashing costs of the other agents. Indeed, for an agent to be able to elaborate the best response to the other agents' strategies, he/she only needs to know the

duration of each activity chosen by the other agents and the topology of the whole project network. Instead, in order to enforce a certain Nash equilibrium, the project owner needs to have complete information on all the agents' parameters. Hence, our model falls in the category of *direct protocols* according to [Kress et al. 2018], where the agents publicly disclose their job durations.

The project scheduling problem presented in this paper is an extension of the classical project time/cost trade-off problem [Phillips and Dessouky 1977]. With respect to the classification of scheduling problems introduced in [Brucker et al 1999], our setting builds on a variant of $MPS|prec|C_{max}$ problem where each activity duration can be viewed as a different *mode* to carry out the activity, the capacity of resources is disregarded, and the reward is assumed as a non-renewable resource (see Section 4 of [Brucker et al 1999]). The additional problem characteristics are the presence of agents and milestones.

This paper revisits the models presented in previous papers, in which milestones were not considered. In [Agnetis et al., 2015], it was shown that finding the minimum makespan Nash equilibrium is in general NP-hard. A large ILP formulation was proposed in [Briand et al. 2017] to compute the minimum-makespan Nash equilibrium. The contribution of this paper is twofold:

(i) On the methodological side, we propose a new algorithmic approach for finding the minimum-makespan equilibrium, based on a compact master formulation and a lazy-constraint generation technique. When no milestones are considered, we compare the efficiency of this algorithm with the solution of the ILP formulation described in [Briand et al. 2017].

(ii) From a managerial viewpoint, we employ the model and algorithm presented in the paper to investigate the role of milestones – and how they should be set up – to drive the project towards the owner's goal.

The remainder of the paper is structured as follows. In Section 2, we discuss existing literature on multi-agent project scheduling problems. In Section 3, basic definitions are reviewed, and the problem is formally stated. Complexity issues are discussed in Section 4. In Section 5, we define the key concepts through which an agent can modify its profit in the face of a given strategy. In Section 6, an ILP formulation for the problem is introduced, and the lazy-constraint approach is illustrated in detail. Section 7 illustrates the design of computational experiments, the results, and managerial insights. Finally, in Section 8, our conclusions are drawn.

2. Related literature

Various models have been introduced to describe the problems arising when multiple agents interact in the execution of a complex project. Some models explicitly consider how common limited resources can be allocated to agents. In [Confessore et al. 2007], an auction-type mechanism is used for coordinating resource allocation among agents, each owning a particular project. Specific allocation mechanisms and their theoretical properties are analyzed in [Varakantham and Fu 2017], while

in [Van Eynde 2017] a resource negotiation mechanism is presented inspired by the dynamics of board games.

A few papers propose cooperative game-theoretic approaches to address various project management issues. In the model analyzed in [Brânzei et al 2002], penalties are generated as some activities are delayed. The authors analyze several rules for sharing penalties among the agents. This is also the focus of [Estévez-Fernández 2012], in which general reward functions for project expediting are considered. A cooperative game-theoretic approach is used in [Moradi et al. 2019] to assess the risks related to agent coordination in a multi-agent project scheduling environment. In [Arık et al. 2019] a scenario is studied in which agents may form coalitions to decrease their individual costs in the face of project makespan reduction. They propose a mathematical program to find the best tradeoff between the project owner's and agents' objectives of reducing project makespan and, respectively, maximizing individual profit.

Non-cooperative models have also been proposed. A literature overview of game-theoretic perspectives on machine scheduling problems is provided in [Kress et al. 2018]. The authors concentrate on problems where part of the machine scheduling problem's relevant data is private information of selfish agents. The paper provides a systematic classification scheme extending the $\alpha|\beta|\gamma$ notation. In [Averbakh 2010], two agents perform the same activities of a project, and they get revenue for each activity completed before the other agent. Their strategy consists of deciding the order in which activities should be carried out, and a polynomial algorithm is given for computing Nash equilibria. In the non-cooperative model proposed in [De Ita Luna et al. 2015], the agents' strategies consist of choosing the subset of activities to be performed and their sequence. The authors analyze the problem of finding the best Nash equilibrium and propose a greedy heuristic. A somewhat closer model to ours has been studied in [Bergantiños and Lorenzo 2019]. In their model, the project owner penalizes individual firms when there are delays. They compare the situation in which the penalty applies only when the whole project is delayed and the situation in which individual agents are penalized upon delaying their activities, even if the project is completed on time. While the application of the two models results in different utility for the project owner and the agents, the authors find that project delay is relatively insensitive to this issue.

In this paper, we extend the modeling approach in [Agnetis et al., 2015, Briand et al. 2017]. These papers deal with finding minimum-makespan Nash equilibria in a situation in which agents are rewarded for expediting project completion, but they bear costs to shorten their respective activities. The problem is shown to be strongly NP-hard in [Agnetis et al., 2015], and a large ILP formulation is proposed in [Briand et al. 2017] for its solution. In [Agnetis et al., 2019], an analysis of the makespan values corresponding to Nash equilibria shows that the increase of makespan values related to Nash equilibria with respect to a reference situation where stability is not required (*price of stability*) is typically small. This observation advocates the effectiveness, for the project owner, of issuing binding agreements based on the best (i.e., minimum-makespan) Nash equilibria.

3. Multi-agent project scheduling with milestones

In order to formally state the problem addressed in this paper, called Multi-Agent Project Scheduling problem with Milestones (MAPSM), we must introduce a number of definitions. We assume basic knowledge of the classical, single-agent project time/cost trade-off problem (see, for example, [Phillips and Dessouky 1977]).

- **Project network.** $G = (N, E)$ is an activity-on-arc network. N is the set of nodes ($N = \{0, \ldots, r\}$) and E is the set of arcs. Arcs correspond to project activities, nodes specify finish-start precedence relations. Nodes 0 and r represent the project beginning and end, respectively. In what follows, n denote the number of real (i.e., non-dummy) activities.

- **Agents and strategies.** The activities are partitioned among a agents, denoted as $A_1, \ldots, A_a$. The set of activities belonging to agent A_u is denoted by E_u. An agent has total control over its activities only. The *individual strategy* of agent A_u, denoted by S_u, consists in deciding the duration of each activity $(i, j) \in E_u$, denoted by p_{ij} and is assumed to be integer. A *strategy* S is a collection of a individual strategies S_u, one for each agent. Given a strategy S, the *project makespan* $D(S)$ is the length of the longest path from node 0 to node r. Given a strategy S, we say that a node $i \in N$ *is reached at* t_i if all activities entering node i are completed at time t_i. Note that $t_r = D(S)$.

- **Normal and crash durations.** For any activity $(i, j) \in E$, it must hold $\underline{p}_{ij} \leq p_{ij} \leq \overline{p}_{ij}$. The values $\underline{p}_{ij}$ and $\overline{p}_{ij}$ are called *crash* and *normal* duration of (i, j) respectively. Symbols $\underline{S}$ and $\overline{S}$ denote strategies in which all activities have crash and, respectively, normal duration, while $\underline{D}$ and $\overline{D}$ indicate the corresponding makespan values. For each node $i \in N$, t_i^{min} and t_i^{max} denote the times at which i is reached in $\underline{S}$ and $\overline{S}$ respectively.

- **Crashing costs.** For each activity (i, j), c_{ij} is its unit crashing cost. So, if the duration of (i, j) is p_{ij}, the agent incurs the cost $c_{ij}(\overline{p}_{ij} - p_{ij})$.

- **Reward and sharing ratios**. We denote by $\pi > 0$ the reward endowed by the project owner to the agents for each day the project makespan is decreased with respect to the normal makespan. So, the *total reward* distributed by the project owner for shortening the makespan from $\overline{D}$ to D is $\pi(\overline{D} - D)$. Agent A_u receives a given, fixed *share* w_u of the total reward ($w_u \geq 0$, $A_u \in \mathcal{A}$ and $\sum_{A_u \in \mathcal{A}} w_u = 1$). Therefore, for a given strategy S, agent A_u receives $w_u \pi(\overline{D} - D(S))$.

- **Milestones and penalties.** We let $N_m \subseteq N$ denote the set of *milestones*. For each milestone $m \in N_m$, a *due date* d_m is specified. Given a certain strategy S, a milestone m *is reached* when all activities corresponding to the arcs entering milestone m are completed. The time $t_m(S)$ (or simply t_m) at which the milestone m is reached is called the *makespan of milestone* m. Then, $T_m = \max\{0, t_m - d_m\}$ is the *tardiness* of milestone m. If $T_m > 0$, we say that the milestone m is *tardy*, while if $t_m - d_m < 0$ it is *early*. We also say that m is *non-tardy* if $t_m - d_m \leq 0$ and it

is *non-early* if $t_m - d_m \geq 0$. A per-day tardiness charge q_{mu} is specified for each milestone m and agent A_u, hence the penalty paid by agent A_u related to milestone m is $q_{mu}T_m$. Without loss of generality, we assume that the project end is always a milestone, i.e., $r \in N_m$. Note that, as long as the current makespan exceeds d_r, for each day the project makespan is reduced each agent A_u receives q_{ru} in addition to $w_u\pi$. Of course, if the project owner does not wish to enforce a specific due date for project completion, $q_{ru} = 0$ for all agents.

- **Agent profit**. Given a strategy S, the *profit* of agent A_u, denoted by $Z_u(S)$, results from the balance between rewards, costs and penalties. Profit $Z_u(S)$ is given by

$$Z_u(S) = w_u\pi(\bar{D} - D(S)) - \sum_{(i,j)\in E_u} c_{ij}(\bar{p}_{ij} - p_{ij}) - \sum_{m\in M} q_{mu}T_m.$$

- **Nash equilibria**. A strategy $S = \{S_1, \ldots, S_a\}$ is a *Nash equilibrium* if for all agents A_u and each strategy $S'_u \neq S_u$,

$$Z_u(S_{-u}, S_u) \geq Z_u(S_{-u}, S'_u). \tag{1}$$

where S_{-u} denotes the individual strategies of all agents other than A_u. If S is a Nash equilibrium, no agent A_u has interest in changing its strategy from S_u to S'_u, as long as other agents do not change theirs, and therefore a Nash equilibrium is a stable strategy.

We can now formulate the problem addressed in this paper as follows:

Multi-Agent Project Scheduling problem with Milestones (MAPSM): Given a project network, the set of agents, the set of milestones, cost and reward information for each agent, find, among Nash equilibria, a strategy having the minimum project makespan.

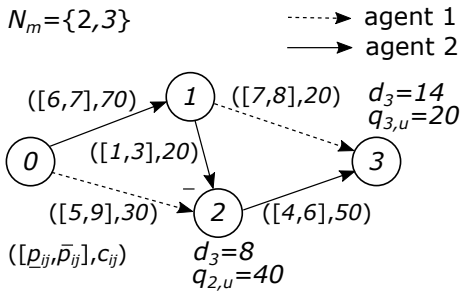

Figure 1: An example of MAPSM.

Example 1. *Fig. 1 illustrates a small instance of* MAPSM *with five activities* $\{(0,1),(0,2),(1,2),(1,3),(2,3)\}$*, and two agents. Agent* A_1 *owns activities* $(0,2)$ *and* $(1,3)$*, agent* A_2 *owns* $(0,1),(1,2)$ *and* $(2,3)$*. Each activity is characterized by* $([\underline{p}_{ij}, \bar{p}_{ij}], c_{ij})$*. The per-day reward from the project owner is* $\pi = 120$*, which is shared equally among the agents* ($w_u = 0.5$)*. The project assumes two milestones* $N_m = \{2,3\}$*. The due date of milestone* 2 *is* 8*, and its per time unit penalty is* 40 *for each agent* ($q_{21} = q_{22} = 40$)*, the due date of milestone 3 is* 14*, and its penalty is* 20 *for each agent* ($q_{31} = q_{32} = 20$)*.*

Under the normal strategy (i.e., $\bar{S} = (7, 9, 3, 8, 6)$), the project makespan is 16 and the profit of the agents is negative ($Z_1 = Z_2 = -120$), since they are penalized for tardy milestones 2 and 3. An example of a Nash equilibria is $S' = (7, 8, 1, 7, 6)$. In this case, the project makespan is 14, and the profits of A_1 and A_2 are $Z_1 = 70$ and $Z_2 = 80$ respectively. No agent has an incentive to deviate from S'. On the other hand, solution $S'' = (6, 8, 2, 7, 5)$, having makespan equal to 13 and profits $Z_1 = 140$, $Z_2 = 40$ is not a Nash equilibria since agent 2 can increase the duration of $(0, 1)$ and $(2, 3)$ and decrease the duration of $(1, 2)$. This increases the makespan by 1 but it also increases the profit of A_2 by 40. Unfortunately, this will not please A_1 whose profit will be reduced by $\pi w_1 = 60$.

4. Problem complexity and the existence of a Nash equilibrium

MAPSM is NP-hard in the strong sense, even when $N_m = \emptyset$ [Agnetis et al., 2015]. Note that if we only aim at finding *any* strategy which is a Nash equilibrium the complexity is different.

Theorem 1. *Given a project network, the set of agents, the set of milestones, cost, and reward information for each agent, a Nash equilibrium can be found in polynomial time.*

Proof. In order to find a Nash equilibrium, one can search for a strategy S_L which is lexicographically optimal for the various agents, i.e., given an agents' numbering, a strategy that (i) maximizes the profit for agent A_1, (ii) among all strategies which are optimal for agent A_1, maximizes the profit of agent A_2; (iii) among the strategies satisfying the two above conditions, maximizes the profit of agent A_3, and so on. Strategy S_L is clearly a Nash equilibrium since a better strategy exists for, let say, agent A_k only if at least another agent A_i with $i < k$ gives up part of its profit with respect to S_L. Obviously, a different numbering of the agents may yield a different strategy, but still a Nash equilibrium. Strategy S_L can be found by solving a linear program, as reported in Appendix A.1. □

A consequence of Theorem 1 is the following corollary.

Corollary 1. *Every instance of problem* MAPSM *has at least one Nash equilibrium.*

5. Decreasing and increasing cuts

This section elaborates on how an agent can improve its profit when a certain strategy S is given. The following definitions are needed, which generalize definitions given in [Agnetis et al., 2015].

- **Non-poor strategy**. A strategy S is a *non-poor strategy* if no agent A_u can increase its profit by modifying the processing times of its activities without affecting the project makespan and without increasing the tardiness of any milestone. Formally, S is *non-poor* if for no agent A_u an individual strategy S'_u exists such that $Z_u(S) < Z_u(S'_u)$, $D(S') = D(S)$ and $t_m(S') \leq t_m(S)$ for all $m \in N_m$, with $S' = (S_{-u}, S'_u)$. A Nash equilibrium is always non-poor.

- **Critical graph** $\mathcal{G}(S)$. Given a strategy S, the longest paths from node 0 to each milestone m on G are the critical paths. An activity is critical if it belongs to at least one critical path. We let

$A(S)$ denote the set of all critical activities, and call *critical graph* the graph $\mathcal{G}(S) = (N, A(S))$ consisting of all critical activities.

If S is non-poor, the only way for A_u to increase its profit is to change the duration of activities in E_u, modifying either the project makespan, or the tardiness of some milestone(s), or both. Sensible modifications of A_u to a strategy S can be characterized in terms of *cuts* in the critical graph $\mathcal{G}(S)$.

- **Cut in G.** Given a partition $(X, N \setminus X)$ of the set of nodes N such that $0 \in X$ and at least one milestone m is such that $m \in N \setminus X$, a *cut* $\omega(X)$ of G is the subset of arcs across X and $N \setminus X$. The arcs $(i, j) \in \omega(X)$ with $i \in X$ and $j \in N \setminus X$ are called *forward arcs*, denoted by $\omega^+(X)$. The arcs $(i, j) \in \omega(X)$ with $i \in N \setminus X$ and $j \in X$ are called *backward arcs*, denoted by $\omega^-(X)$, and $\omega(X) = \omega^+(X) \cup \omega^-(X)$.

- **Decreasing cut for agent A_u.** Given a strategy S and the corresponding critical graph $\mathcal{G}(S)$, a *decreasing cut for agent* A_u is a cut $\omega^{(u)}_{dec}$ on $\mathcal{G}(S)$ such that *all* forward arcs belong to A_u and the duration of each of them can be decreased by one unit. We denote forward and backward arcs as $\omega^{+(u)}_{dec}$ and $\omega^{-(u)}_{dec}$.

- **Increasing cut for agent A_u.** Given a strategy S and the corresponding critical graph $\mathcal{G}(S)$, an *increasing cut for agent* A_u is a cut $\omega^{(u)}_{inc}$ on $\mathcal{G}(S)$ such that *at least one* forward activity belongs to A_u and its duration can be increased by one time unit, while *all* backward activities belong to A_u and the duration of each of them can be decreased by one unit. We denote forward and backward arcs as $\omega^{+(u)}_{inc}$ and $\omega^{-(u)}_{inc}$ respectively.

We say that we *apply a decreasing cut* $\omega^{(u)}_{dec}(X)$ to mean that we decrease (by one time unit) the duration of all activities in $\omega^{+(u)}_{dec}(X)$, and increase (by one time unit) the duration of all activities in $\omega^{-(u)}_{dec}(X)$ which can be extended. Decreasing (by one time unit) the duration of all activities in $\omega^{+(u)}_{dec}(X)$, we decrease by *at least* one time unit the makespan of all milestones which belong to $N \setminus X$. In particular, if *all* backward arcs can be increased by one time unit, then the makespans of all milestones in $N \setminus X$ are decreased by exactly one time unit. (Notice that if $r \in X$, the project makespan is not affected by such operation.)

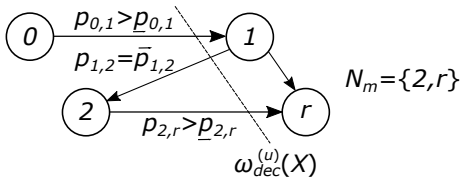

Figure 2: Decreasing cut decreasing the makespan by 2.

However, suppose that all the paths leading to a certain milestone m in $N \setminus X$ have two (or more) arcs in $\omega^{+(u)}_{dec}(X)$, and *all* such paths have at least one arc in $\omega^{-(u)}_{dec}(X)$ which cannot be increased (either because such an arc has normal duration or because it does not belong to A_u). An example of such a scenario is illustrated in Fig. 2. In this case, t_r is decreased by at least two time units, and

the backward arc becomes non-critical. Moreover, in this case certain arcs $(i,j) \in \omega_{dec}^{+(u)}(X)$ are such that no path fully contained in X exists from the initial node 0 to i. Let $\tilde{U} \subseteq X$ be the set of such nodes i. If some $i \in \tilde{U}$ is a milestone, its makespan is also affected by the cut. In the example in Fig. 2, t_2 is affected by the application of the decreasing cut, even though $2 \in X$.

When a decreasing cut $\omega_{dec}^{(u)}(X)$ is applied, the agent bears a crashing cost related to shortening forward activities, but also savings due to (i) extending backward activities, (ii) possibly reducing the tardiness of some milestone, and (iii) reducing the overall project makespan. Hence, we can associate with a decreasing cut $\omega_{dec}^{(u)}(X)$ a *unit cost* $W_{dec}^{(u)}(X)$ defined as:

$$W_{dec}^{(u)}(X) = \sum_{(i,j)\in\omega_{dec}^{+(u)}(X)} c_{ij} - \sum_{(i,j)\in\omega_{dec}^{-(u)}(X), p_{ij}<\bar{p}_{ij}} c_{ij} - \sum_{m\in N\setminus X, m \text{ is tardy}} q_m - \pi w_u. \tag{2}$$

In general, (2) overestimates the real cost, since, as we already observed, a unit decrease in the activities of $\omega_{dec}^{+(u)}(X)$ may result in a multiple decrease of the makespan of some milestone(s) in $N \setminus X$, and possible milestones in X are not accounted for in (2). So, if we let $\tilde{W}_{dec}^{(u)}(X)$ be the *real* cost related to applying the cut $\omega^{(u)}(X)$, it may happen that $\tilde{W}_{dec}^{(u)}(X) < W_{dec}^{(u)}(X)$.

Nonetheless, the next theorem shows that decreasing cuts that overestimate the real cost can be disregarded when searching for a decreasing cut that minimizes $W_{dec}^{(u)}(X)$.

Theorem 2. *If $\omega_{dec}^{(u)}(X^*)$ minimizes $W_{dec}^{(u)}(X)$, then $W_{dec}^{(u)}(X^*) = \tilde{W}_{dec}^{(u)}(X^*)$.*

Proof. See Appendix A.2. □

A symmetric discussion holds with respect to increasing cuts. We say that we *apply an increasing cut* $\omega_{inc}^{(u)}(X)$ to mean that we increase (by one time unit) the duration of all forward activities belonging to A_u which are not at normal duration (by definition of increasing cut, there must be at least one such activity) and decrease by one time unit *all* backward activities (they all belong to A_u by definition).

By extending (by one time unit) all forward activities of $\omega_{inc}^{(u)}(X)$ that are not at normal duration, A_u gets a saving. However, to avoid multiple extensions of the makespans of the milestones in $N \setminus X$, *all* backward activities must be crashed. The benefit from activity extension is therefore balanced by (i) the additional crashing cost of backward activities, (ii) the possibly increased tardiness of milestones in $N \setminus X$, and (iii) the missed reward due to project makespan increase. So, one can associate with an increasing cut $\omega_{inc}^{(u)}(X)$ the following *unit saving* $W_{inc}^{(u)}(X)$:

$$W_{inc}^{(u)}(X) = \sum_{(i,j)\in\omega_{inc}^{+(u)}(X)} c_{ij} - \sum_{(i,j)\in\omega_{inc}^{-(u)}(X)} c_{ij} - \sum_{m\in N\setminus X, m \text{ is nonearly}} q_m - \pi w_u. \tag{3}$$

Symmetrically to Equation (2), it may happen that (3) underestimates the real saving. In fact, the definition of increasing cut only requires that at least one forward activity (call it (i,j)) can be increased. The set $\omega_{inc}^{+(u)}(X)$ may contain other activities, but these may be at normal duration or belong to other agents, so they cannot be indeed increased. In conclusion, increasing the duration of activity (i,j) may not affect the makespans of all milestones in $N \setminus X$. So, if we call $\tilde{W}_{inc}^{(u)}(X)$ the

real saving achieved by A_u extending the activities of $\omega_{inc}^{+(u)}(X)$ (and crashing by one time unit all the activities of $\omega_{inc}^{-(u)}(X)$ which can be crashed), it may happen that $\tilde{W}_{inc}^{(u)}(X) > W_{inc}^{(u)}(X)$.

Nonetheless, a symmetric result to Theorem 2 can be established.

Theorem 3. *If $\omega_{inc}^{(u)}(X^*)$ maximizes $W_{inc}^{(u)}(X)$, then $W_{inc}^{(u)}(X^*) = \tilde{W}_{inc}^{(u)}(X^*)$.*

Proof. See Appendix A.3. □

Now, given a strategy S, for a decreasing cut $\omega_{dec}(X)$ to be *profitable* to a certain agent, one must have that $W_{dec}(X) < 0$ in (2), while symmetrically an increasing cut $\omega_{inc}(X)$ is profitable if $W_{inc}(X) > 0$ in (3). This leads to the following characterization of Nash equilibria (see [Agnetis et al., 2015] for formal details).

Theorem 4. *Given a strategy S, S is a Nash equilibrium if an only if, for each agent $A_u \in \mathcal{A}$, for all decreasing cuts $\omega_{dec}^{(u)}(X)$, $W_{dec}^{(u)}(X) \geq 0$ and for all increasing cuts $\omega_{inc}^{(u)}(X)$, $W_{inc}^{(u)}(X) \leq 0$.*

The above characterization of Nash equilibrium is used in our algorithm described in the following section.

6. ILP formulation with lazy constraints

An ILP formulation exploiting the duality between the minimum value of the flow and the maximum cut capacity in a residual network was used to solve MAPSM without milestones in [Briand et al. 2017]. A disadvantage of this approach is that the ILP model size quickly grows with the number of activities n and even more with the number of agents a. Since the problem with milestones would require an even larger ILP formulation, we propose a different approach. We define a relatively small *master problem* formulation in which constraints related to increasing and decreasing cuts are progressively generated in a lazy-constraint fashion by solving an appropriate *subproblem*. The main advantage is a lower memory requirement and a higher performance, as it is shown by experiments in Section 7.

In this section, we first give a conceptual formulation of MAPSM and then show how it can be solved through a lazy-constraint generation approach (see [Dantzig et al. 1959]). In this formulation, s_{ij} denotes the slack time of activity (i, j).

$$\min \left(t_r + \frac{\sum_{\forall (i,j)\in E} c_{ij}\left(\overline{p}_{ij} - p_{ij}\right) + \sum_{\forall m \in N_m} \sum_{\forall u \in \mathcal{A}} q_{mu} T_m}{1 + \sum_{\forall (i,j)\in E} c_{ij}\left(\overline{p}_{ij} - \underline{p}_{ij}\right) + \sum_{\forall m \in N_m} \sum_{\forall u \in \mathcal{A}} q_{mu}\left(t_m^{max} - d_m\right)} \right) \tag{4}$$

s.t.

$$t_j - t_i - p_{ij} - s_{ij} = 0 \qquad \forall (i,j) \in E \tag{5}$$

$$\underline{p}_{ij} \leq p_{ij} \leq \overline{p}_{ij} \qquad \forall (i,j) \in E \tag{6}$$

$$t_0 = 0 \tag{7}$$

$$t_m - d_m \leq T_m \qquad \forall m \in N_m \tag{8}$$

$$W_{dec}^{(u)}(X) \geq 0 \qquad \forall \omega_{dec}^{(u)}(X), \forall A_u \in \mathcal{A} \tag{9}$$

$$W_{inc}^{(u)}(X) \leq 0 \qquad \forall \omega_{inc}^{(u)}(X), \forall A_u \in \mathcal{A} \tag{10}$$

$$t_i, T_m, p_{ij} \in \mathbb{Z}_{\geq 0}, \quad s_{ij} \in \mathbb{R}_{\geq 0} \tag{11}$$

The objective function (4) has two parts. The first one minimizes project makespan t_r. The second part ensures that the optimal solution is non-poor by minimizing the overall cost of all agents, i.e., overall crashing cost and tardiness penalty. As t_r is an integer value, and the second term is always smaller than 1, the second term does not affect the project makespan. Constraints (5) and (6) define precedence constraints and processing time ranges for each activity. The project starts at time 0 (constraint (7)). Constraints (8) define milestone tardiness T_m. Constraints (9) and (10) express the fact that there must not be any profitable decreasing or increasing cut, and hence the strategy is a Nash equilibrium.

Constraints (9) and (10) cannot be directly written in linear terms, as a cut is increasing or decreasing (or none) depending on the strategy, and even for a given strategy, there are in general exponentially many increasing or decreasing cuts. We solve this issue by defining the master problem, described in the next subsection, that disregards constraints (9) and (10). These constraints are then handled by a lazy-constraint generation scheme, explained later on in Subsection 6.2.

6.1. Master problem

The master problem formulation is obtained by retaining constraints (4)–(8) and adding a new set of constraints. Such new constraints use a new set of variables carrying the relevant information with respect to the existence of a profitable cut, i.e., (*i*) for each activity, whether the activity is critical, and whether it has crash, normal or intermediate duration, and (*ii*) for each milestone, whether it is early, non-early, tardy or non-tardy.

We introduce the new binary variables x_{ij},y_{ij},z_{ij},τ_m,τ'_m which all refer to a certain strategy $S = (S_1, \ldots, S_a)$:

$x_{ij} = 1$ if $p_{ij} < \bar{p}_{ij}$, i.e., if $(i,j) \in A$ can be extended, and 0 otherwise;

$y_{ij} = 1$ if $p_{ij} > \underline{p}_{ij}$, i.e., if $(i,j) \in A$ can be crashed, and 0 otherwise;

$z_{ij} = 1$ if $(i,j) \in A$ is critical, and 0 otherwise;

$\tau_m = 1$ if milestone $m \in N_m$ is non-early and 0 otherwise;

$\tau'_m = 1$ if milestone $m \in N_m$ is tardy and 0 otherwise.

Hence, the following constraints are added to (4)–(8):

$$\epsilon - z_{ij} \leq s_{ij} \leq \bar{s}_{ij}\,(1 - z_{ij}) \qquad \forall (i,j) \in E \tag{12}$$

$$z_{ij} \leq \sum_{\forall (k,i) \in E} z_{ki} \qquad \forall (i,j) \in E : i > 1 \tag{13}$$

$$z_{ij} \leq \sum_{\forall (j,l) \in E} z_{jl} \qquad \forall (i,j) \in E : j < n, j \notin N_m \tag{14}$$

$$\sum_{\forall (i,m) \in E} z_{im} \geq 1 \qquad \forall m \in N_m \tag{15}$$

$$1 \leq \sum_{\forall (1,i) \in E} z_{1i} \tag{16}$$

$$t_m \leq d_m - 1 + (t_m^{max} - t_m^{min} + 1)\tau_m \qquad \forall m \in N_m \qquad (17)$$
$$t_m \geq d_m - (t_m^{max} - t_m^{min})(1 - \tau_m) \qquad \forall m \in N_m \qquad (18)$$
$$t_m \leq d_m + (t_m^{max} - t_m^{min})\tau'_m \qquad \forall m \in N_m \qquad (19)$$
$$t_m \geq d_m + 1 - (t_m^{max} - t_m^{min} + 1)(1 - \tau'_m) \qquad \forall m \in N_m \qquad (20)$$

$$x_{ij} \leq \left(\overline{p}_{ij} - p_{ij}\right) \leq \left(\overline{p}_{ij} - \underline{p}_{ij}\right) x_{ij} \qquad \forall (i,j) \in E \qquad (21)$$
$$y_{ij} \leq \left(p_{ij} - \underline{p}_{ij}\right) \leq \left(\overline{p}_{ij} - \underline{p}_{ij}\right) y_{ij} \qquad \forall (i,j) \in E \qquad (22)$$
$$z_{ij} \geq x_{ij} \qquad \forall (i,j) \in E \qquad (23)$$
$$x_{ij}, y_{ij}, z_{ij}, \tau_m, \tau'_m \in \{0,1\}$$

Constraints (12) specify that if an activity (i,j) has a nonzero slack variable s_{ij}, then it cannot be critical, and that if an activity (i,j) is critical, then $s_{ij} = 0$. Constraints (13) and (14) ensure continuity of critical paths in the graph. Namely, if an activity (i,j) is critical, then (13) implies that at least one activity entering node i is also critical. If activity (i,j) is critical and node j is not a milestone, then at least one activity leaving node j is critical, due to (14). Constraints (15) imply that at least one critical path exists from 0 to every milestone, while constraint (16) implies that at least one critical path starts from node 0. The meaning of variables τ_m, τ'_m is specified by constraints (17) - (20), that of variables x_{ij} and y_{ij} by constraints (21) and (22) respectively. Finally, we can assume that if an activity is not critical, it cannot be increased, since otherwise it would be convenient for the agent to actually extend the activity, and the strategy would not be non-poor. This is enforced by constraints (23).

The values of the variables x_{ij}, y_{ij}, z_{ij} characterize a cut on $\mathcal{G}(S)$ in terms of being increasing, decreasing or none for a certain agent A_u. In particular, given a strategy S, a cut $\omega^{(u)}(X)$ is increasing if, for all $(i,j) \in \omega^{+(u)}(X)$ such that $z_{ij} = 1$, one has $y_{ij} = 1$, while a cut $\omega^{(u)}(X)$ is decreasing if there exists an arc $(i,j) \in \omega^{+(u)}(X)$ such that $z_{ij} = 1$ and $x_{ij} = 1$, while $y_{ij} = 1$ for all $(i,j) \in \omega^{-(u)}(X)$. The profitability of a decresing/increasing cut also depends on the status of the various milestones, which is specified by variables τ_m and τ'_m.

6.2. Lazy constraints generation scheme

Constraints (5)–(8) and (12)–(23) with the objective function (4) form the master problem. The optimal solution of the master problem may not be a Nash equilibrium. Therefore, in order to solve MAPSM, we dynamically add new constraints to the master problem, using the lazy constraints generation mechanism illustrated in Alg. 1. Whenever the ILP solver finds a new best integer solution S of the master problem, we check for every agent whether there exist profitable decreasing or increasing cuts. This check is done by solving the subproblem, as described later in Section 6.3. The ILP solver automatically launches this mechanism as a user-defined *call-back function*. For each agent, our algorithm checks if there is a profitable increasing or decreasing cut. If the subproblem

Algorithm 1: Procedure generating the lazy constraints.

```
Function IntegerSolutionFoundCallback(S)
    foreach A_u ∈ 𝒜 do
        foreach cutType ∈ {inc, dec} do
            /* Search for a profitable cut X* (see sections 6.2 and 6.3).   */
            X* = Subproblem(cutType, u, S);
            if X* is profitable for agent A_u then
                Use X* to add an new lazy constraint (LC) to the master model;
                /* see Section 6.4                                          */
            end
        end
    end
end
```

finds a profitable cut X^*, the cut is used to define a new lazy constraint (LC), which is subsequently added to the master model. The new constraint makes solution S infeasible, and the ILP solver searches for another solution of the master problem until the best solution S^* is found. On the other hand, if there are no profitable cuts, S is a new incumbent Nash equilibrium.

Preliminary experiments revealed that the most effective lazy constraints are obtained for relatively *small* cuts, i.e., cuts having few (either forward or backward) arcs. This is because the lazy constraint corresponding to a small cut can cut off many more non-Nash solutions. For this reason, as explained in more detail in Section 6.3, for each agent we also seek for the profitable decreasing or increasing cut with the minimum number of arcs, and add the corresponding lazy constraint to the master problem.

Another experiment-based observation is related to the number of generated lazy constraints. In the case of very large instances, the number of generated lazy constraints can be huge. This fact enormously increases the size of the MILP model, which harms the overall algorithm performance, as shown in Section 7.2. Therefore, it is better for large instances to generate only *one* lazy constraint per solution that is not a Nash equilibrium. If there are many possible lazy constraints, we add the one related to a cut with the minimum number of forward and backward arcs.

Hence, in order to implement the scheme in Alg.1, we must specify two key issues related to the subproblem, namely: (i) how to find a profitable cut, and (ii) how to specify a lazy constraint to be added to (5)–(23). These two aspects are detailed in the next subsections.

6.3. Computing profitable decreasing and increasing cuts - subproblems

In order to detect a profitable cut, it is convenient to introduce a *residual network*, and express the search for a profitable cut in terms of finding a suitable cut on such a network. To this aim, given a strategy S and an agent A_u, we consider a network $\mathcal{N}_u(S)$ having the same arc and node sets of

G, but in which each arc (i, j) has associated lower and upper capacities ℓ_{ij} and u_{ij} respectively, as specified in Table 1.

The table allows representing decreasing and increasing cuts for agent A_u as cuts in a classical network flow problem. Lower and upper capacities do not represent only the crashing cost of activities, but also a possibility to increase or decrease their duration, and their impact on the makespan and milestones. For example, the first row of Table 1 defines the situation when an activity is not critical. In this case, it is not important whether its duration can be increased or decreased. This arc has no impact on any milestone or project makespan. Therefore $\ell_{ij} = u_{ij} = 0$. Similarly, if the activity is critical and can be increased but not decreased (row 4) then it can be used as a forward arc in an increasing cut or as a backward arc in a decreasing cut with cost $l_{ij} = c_{ij}$. However, in other cases the entire cut is disregarded, letting $u_{ij} = +\infty$. For more details about this transformation, please see [Agnetis et al., 2019].

Recall (e.g. [Ahuja et al. 1993]) that, given a flow network in which the flow in each arc (i, j) must belong to the interval $[\ell_{ij}, u_{ij}]$, the *capacity* of a cut $\omega(X)$ is defined as

$$k(\omega(X)) = \sum_{(i,j)\in\omega^+(X)} u_{ij} - \sum_{(i,j)\in\omega^-(X)} \ell_{ij},$$

while the *floor* as

$$\phi(\omega(X)) = \sum_{(i,j)\in\omega^+(X)} \ell_{ij} - \sum_{(i,j)\in\omega^-(X)} u_{ij}.$$

Hence, recalling expressions (2) and (3), and in view of Table 1, given a decreasing cut $\omega^{(u)}_{dec}(X)$, we can write its cost as

$$W^{(u)}_{dec}(X) = k(\omega^{(u)}_{dec}(X)) - \sum_{m\in N\setminus X, m \text{ tardy}} q_m - (1-\gamma_r)\pi w_u, \tag{24}$$

where we let $\gamma_r = 1$ if $r \in X$ and $\gamma_r = 0$ if $r \in N \setminus X$. In the latter case, the project makespan is affected by the cut. Similarly, the saving of an increasing cut is:

$$W^{(u)}_{inc}(X) = \phi(\omega^{(u)}_{inc}(X)) - \sum_{m\in N\setminus X, m \text{ nonearly}} q_m - (1-\gamma_r)\pi w_u. \tag{25}$$

Hence, *the most profitable decreasing cut* is the cut for which (24) is minimum, while *the most profitable increasing cut* is the cut for which (25) is maximum.

We now explain in more detail how to compute the most profitable decreasing and increasing cuts.

6.3.1. Decreasing Cuts

Given a non-poor strategy S and an agent A_u, and hence the values in Table 1, as well as the variables τ_m and τ'_m, a profitable decreasing cut is such that $W^{(u)}_{dec}(X)$ is negative. We next formulate the subproblem of finding the decreasing cut minimizing (24), i.e., the most profitable decreasing cut in the residual network $\mathcal{N}_u(S)$.

Let $M^D = \{m \in N_m : \tau'_m = 1\}$ be the set of tardy milestones in S, which can therefore be affected by a decreasing cut. We introduce the following variables. For each arc $(i, j) \in A$, $\alpha_{ij} = 1$ if (i, j) is

$(i,j) \in A_u$	z_{ij}	x_{ij}	y_{ij}	l_{ij}	u_{ij}
True	0	0/1	0/1	0	0
True	1	1	1	c_{ij}	c_{ij}
True	1	0	1	0	c_{ij}
True	1	1	0	c_{ij}	$+\infty$
True	1	0	0	0	$+\infty$
False	0	0/1	0/1	0	0
False	1	0/1	0/1	0	$+\infty$

Table 1: Lower and upper capacities l_{ij}, u_{ij} in $\mathcal{N}_u(S)$

a forward arc of the cut, and $\beta_{ij} = 1$ if it is a backward arc. For each node $i \in N$, we let $\gamma_i = 1$ if $i \in X$ and $\gamma_i = 0$ if $i \in N \setminus X$. Note in particular that $\gamma_r = 0$ denotes that the final node r belongs to $N \setminus X$, in which case the project makespan is affected by the cut. Hence, the problem of finding the most profitable decreasing cut can be formulated as:

$$\min \sum_{(i,j)\in E} \alpha_{ij} u_{ij} - \sum_{(i,j)\in E} \beta_{ij} l_{ij} - \sum_{m \in M^D} q_{mu}(1-\gamma_m) - \pi w_u (1-\gamma_r) \tag{26}$$

s.t.

$$\alpha_{ij} - \beta_{ij} = \gamma_i - \gamma_j \qquad \forall (i,j) \in E \tag{27}$$

$$\gamma_0 = 1 \tag{28}$$

$$\textstyle\sum_{m \in N_m} \gamma_m \leq |N_m| - 1 \tag{29}$$

$$\alpha_{ij}, \beta_{ij}, \gamma_i \in \{0,1\}$$

Constraints (27) imply that if $(i,j) \in \omega_{dec}^{+(u)}(X)$, then node i is on the left and j is on the right side of the cut, and vice versa if $(i,j) \in \omega_{dec}^{-(u)}(X)$. (Note that since $u_{ij} \geq \ell_{ij}$, with no loss of generality we can assume that when i and j are on the same side of the cut, both α_{ij} and β_{ij} are zero.) Node 0 is always on the left side of the cut due to constraint (28). Constraint (29) specifies that there must always be at least one milestone on the right side of the cut.

The problem (26)–(29) is formulated as an ILP. An alternative approach to solving (26)–(29) is to solve $|N_m|$ instances of a problem obtained from (26)–(29) by imposing, in turn, that a certain milestone m must lie on the right hand side of the cut, i.e., replacing constraint (29) with the constraint $\gamma_m = 0$:

$$\min \sum_{(i,j)\in E} \alpha_{ij} u_{ij} - \sum_{(i,j)\in E} \beta_{ij} l_{ij} - \sum_{m \in M^D} q_{mu}(1-\gamma_m) - \pi w_u (1-\gamma_p) \tag{30}$$

s.t.

$$\alpha_{ij} - \beta_{ij} = \gamma_i - \gamma_j \qquad \forall (i,j) \in E \tag{31}$$

$$\gamma_0 = 1 \tag{32}$$

$$\gamma_m = 0 \tag{33}$$

$$\alpha_{ij} \geq 0, \beta_{ij} \geq 0, \gamma_i \geq 0$$

Notice that as the coefficient matrix of problem (30)–(33) is the incidence matrix of the project network, it is totally unimodular, and hence we could relax integrality constraints on the variables. So a minimum cost decreasing cut can be found by solving $|N_m|$ linear programs (showing therefore that (26)–(29) is polynomially solvable.) However, in the solution procedure we directly solve the ILP (26)–(29), as preliminary runs showed that this is more efficient than solving many LPs (30)–(33).

As it was mentioned in the previous section, preliminary experiments pointed out that the most efficient constraints are typically related to cuts having a low number of arcs. The reason is that a constraint covering fewer activities can cut off larger solution space. Based on this empirical observation, we introduce the following ILP.

$$\min \sum_{(i,j)\in E} (\alpha_{ij} + \beta_{ij}) \tag{34}$$

s.t.

$$\alpha_{ij} - \beta_{ij} = \gamma_i - \gamma_j \qquad \forall (i,j) \in E \tag{35}$$

$$\sum_{(i,j)\in E} \alpha_{ij} u_{ij} - \sum_{(i,j)\in E} \beta_{ij} l_{ij} - \sum_{m \in M^D} q_{mu}(1-\gamma_m) - \pi w_u (1-\gamma_r) < 0 \tag{36}$$

$$\gamma_0 = 1 \tag{37}$$

$$\textstyle\sum_{m\in N_m} \gamma_m \leq |N_m| - 1 \tag{38}$$

$$\alpha_{ij}, \beta_{ij}, \gamma_i \in \{0,1\}$$

Problem (34)–(38) has been obtained from (26)–(29) replacing the objective function (maximization of the profit of a decreasing cut for agent A_u) with the minimization of the total number of arcs in the cut. Note that constraints (36) ensure that the cut is a profitable decreasing cut.

Problem (34)–(38) is used to generate profitable decreasing cuts as long as the application of the found cut decreases the project makespan by one unit. When this condition is not satisfied, we switch to finding the most profitable decreasing cut, i.e., we generate cuts by solving the formulation (26)–(29).

Once a profitable decreasing cut is selected, we define the following sets, which are used in the definition of new lazy constraint LC_k^{dec} (see Section 6.4).

$F_k = \{(i,j) \in E : \alpha_{ij} = 1 \wedge z_{i,j} = 1\}$ is the set of critical forward arcs,

$B_k = \{(i,j) \in E : \beta_{ij} = 1 \wedge z_{i,j} = 1\}$ is the set of critical backward arcs,

$C_k = \{(i,j) \in E : \alpha_{ij} = 1 \wedge z_{i,j} = 0\}$ is the set of non-critical forward arcs,

$D_k = \{m \in N_m : \gamma_m = 0 \wedge \tau'_m = 1\}$ is the set of tardy milestones affected by the cut.

6.3.2. Increasing Cuts

The problem of finding profitable increasing cuts can be addressed through a very similar approach. This time the objective is to find the increasing cut that maximizes the saving. Given the strategy S, we define $M^I = \{m \in N_m : \tau_m = 1\}$ to be the set of *non-early* milestones, which can therefore be

affected by the cut. Notice that if for a milestone $m \in N \setminus X$ one has $t_m = d_m$, the milestone becomes tardy when the cut is applied. The meaning of variables $\alpha_{ij}, \beta_{ij}, \gamma_i$ as well as the constraints are the same as for (26)–(29). This time, the cut maximizing the saving is profitable when the objective function is positive.

$$\max \sum_{(i,j)\in E} \alpha_{ij} l_{ij} - \sum_{(i,j)\in E} \beta_{ij} u_{ij} - \sum_{m\in M^I} q_{mu}(1-\gamma_m) - \pi w_u (1-\gamma_r) \tag{39}$$

s.t.

$$\alpha_{ij} - \beta_{ij} = \gamma_i - \gamma_j \qquad \forall (i,j) \in E \tag{40}$$

$$\gamma_0 = 1 \tag{41}$$

$$\textstyle\sum_{m\in N_m} \gamma_m \leq |N_m| - 1 \tag{42}$$

$$\alpha_{ij}, \beta_{ij}, \gamma_i \in \{0,1\}$$

As before, an alternative approach to solving (39)–(42) would be to solve $|N_m|$ instances of the LP similar to (30)–(33).

In the same way as for the decreasing cuts, the algorithm searches first for profitable cuts having the minimum number of arcs, which can be found solving the problem:

$$\min \sum_{(i,j)\in E} (\alpha_{ij} + \beta_{ij}) \tag{43}$$

s.t.

$$\alpha_{ij} - \beta_{ij} = \gamma_i - \gamma_j \qquad \forall (i,j) \in E \tag{44}$$

$$\sum_{(i,j)\in E} \alpha_{ij} l_{ij} - \sum_{(i,j)\in E} \beta_{ij} u_{ij} - \sum_{m\in M^I} q_{mu}(1-\gamma_m) - \pi w_u (1-\gamma_r) > 0 \tag{45}$$

$$\gamma_0 = 1 \tag{46}$$

$$\textstyle\sum_{m\in N_m} \gamma_m \leq |N_m| - 1 \tag{47}$$

$$\alpha_{ij}, \beta_{ij}, \gamma_i \in \{0,1\}$$

Once a profitable cut is found, we define the following sets which are used in the definition of new lazy constraint LC_k^{dec} (see Section 6.4):

$F_k = \{(i,j) \in E : \alpha_{ij} = 1 \wedge z_{i,j} = 1\}$ is the set of critical forward arcs,

$B_k = \{(i,j) \in E : \beta_{ij} = 1 \wedge z_{i,j} = 1\}$ is the set of critical backward arcs,

$C_k = \{(i,j) \in E : \beta_{ij} = 1 \wedge z_{i,j} = 0\}$ is the set of non-critical backward arcs,

$D_k = \{m \in N_m : \gamma_m = 0 \wedge \tau_m = 0\}$ is the set of early milestones affected by the cut.

6.4. Lazy constraints

In this section, we specify the constraint to add to the master problem formulation (5)–(23), in order to forbid solutions having a particular profitable cut found by the subproblem described in

the previous section. Our intention is to introduce constraints avoiding large coefficients ("big-M"), which would weaken the linear relaxation of the master problem.

6.4.1. Constraints related to decreasing cuts

Given a profitable decreasing cut $\omega_{dec}(X^*)$ for an agent A_u, and hence given the sets F_k, B_k, C_k and D_k defined in Section 6.3.1, the corresponding lazy constraint is defined as follows:

$$\sum_{\forall (i,j) \in F_k} y_{ij} + \sum_{\forall (i,j) \in B_k} x_{ij} \leq |F_k \cup B_k| - 1 + \sum_{\forall (i,j) \in C_k} z_{ij} + \sum_{\forall m \in D_k} (1 - \tau'_m) \tag{48}$$

Notice that, from the definition of the sets F_k, B_k, C_k and D_k, when we plug into (48) the current optimal solution of the master problem, the left-hand-side equals $|F_k \cup B_k|$, while the right-hand-side equals $|F_k \cup B_k| - 1$, so the constraint is indeed violated by $\omega_{dec}(X^*)$. The reason for including variables z_{ij} and τ'_m in the right-hand-side of (48) is the following. While in the current optimal solution of the master problem a forward arc $(i,j) \in C_k$ is non-critical, it might become critical later on. If this occurs, the cost of forward arcs of $\omega_{dec}(X^*)$ becomes higher than in the current situation, and as a result the cut $\omega_{dec}(X^*)$ might be no more profitable for A_u. Hence, if $(i,j) \in C_k$ becomes critical (i.e., if z_{ij} turns to 1), we do not forbid cut $\omega_{dec}(X^*)$ anymore, and the constraint (48) is turned off. Similarly, if a currently tardy milestone m is no more tardy at a later stage of the algorithm, there is no saving connected with anticipating t_m, and hence again, the constraint has to be deactivated. This point is illustrated in the following example.

Example 2. *Fig. 3 shows an example of a residual network. Suppose that in the optimal solution to (26)–(29) one has* $\alpha_{2,5} = \alpha_{3,6} = \alpha_{4,7} = 1$ *and* $\beta_{5,3} = 1$*, so that, letting* $\omega_{dec}(X^*)$ *denote the optimal solution of (26)–(29), one has* $\omega^+_{dec}(X^*) = \{(2,5),(3,6),(4,7)\}$ *and* $\omega^-_{dec}(X^*) = \{(5,3)\}$*. Suppose that arcs* $(2,5)$ *and* $(3,6)$ *are critical (*$z_{2,5} = z_{3,6} = 1$*), while* $(4,7)$ *and* $(5,3)$ *are not (*$z_{4,7} = z_{5,3} = 0$*). Moreover, milestone 8 is tardy (*$\tau'_8 = 1$*), while milestone 9 is early (*$\tau_9 = 0$*). As a consequence,* $F_k = \{(2,5),(3,6)\}, B_k = \emptyset, C_k = \{(4,7)\}$ *and* $D_k = \{8\}$*. The cut* $\omega^+_{dec}(X^*)$ *is a profitable decreasing cut, since the increased crashing costs of forward arcs (=40+50) are outbalanced by the saving on reducing the tardiness of milestone 8 (=100). So, we generate the constraint*

$$x_{2,5} + x_{3,6} \leq 1 + z_{4,7} + (1 - \tau'_8).$$

If arc $(4,7)$ *becomes critical later on, the constraint is turned off, because the cut would no longer be profitable, as* $40 + 20 + 50 - 100 = 10 > 0$*. Similarly, the constraint is turned off if milestone* 8 *becomes non-tardy, as the cost would become* $40 + 50 > 0$*.*

Notice that in specifying the constraints in this way, we are adopting a "conservative" approach. If the crashing cost of activity $(4,7)$ were 5 instead of 20, the cut $\omega^+_{dec}(X^*)$ remains profitable even if $(4,7)$ becomes critical, so it would be correct to continue forbidding it. On the contrary, using constraints (48), it is possible that the same cut will be selected again (though this time F_k would include also arc $(4,7)$). This phenomenon could be avoided using a different expression of lazy

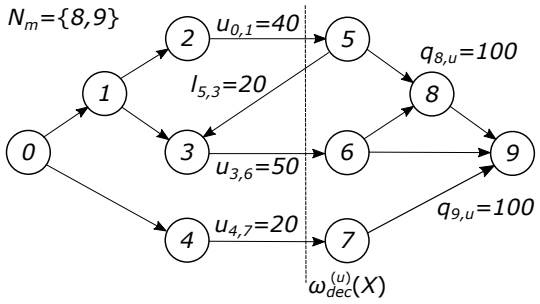

Figure 3: Example of decreasing cut in the residual network of agent A_u.

constraints, in which crashing costs are explicitly considered. However, since this did not often occur in our experiments, we preferred to maintain the constraints (48), whose coefficients are all 0 and 1, as this typically results in a tighter formulation.

6.4.2. Constraints related to increasing cuts

Similar considerations hold for a profitable increasing cut for an agent A_u. Given the increasing cut $\omega_{inc}(X^*)$ that solves (39)–(42) and the corresponding sets F_k, B_k, C_k and D_k, we add to the master problem the following constraint:

$$\sum_{\forall(i,j)\in F_k} x_{ij} + \sum_{\forall(i,j)\in B_k} y_{ij} \leq |F_k \cup B_k| - 1 + \sum_{\forall(i,j)\in C_k} z_{ij} + \sum_{\forall m\in D_k} \tau_m \tag{49}$$

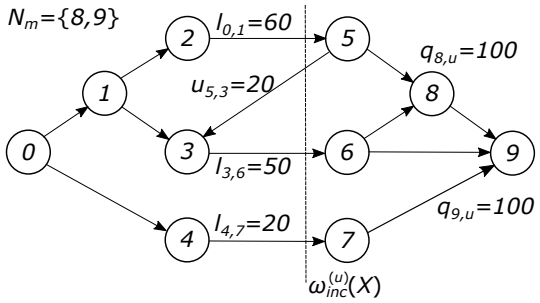

Figure 4: Example of increasing cut in the residual network of agent A_u.

The role of variables z_{ij} for $(i,j) \in C_k$ and τ_m for $m \in D_k$ is similar to that for decreasing cuts. In particular, if a currently non-critical backward arc becomes critical, we turn off the constraint, since the cut may become non profitable, and the same occurs if a currently early milestone becomes non-early. As in the previous case, we illustrate this situation using an example.

Example 3. *Let's assume a residual network illustrated in Fig. 4. Suppose that in* $\omega_{inc}(X^*)$ *one has* $\alpha_{2,5} = \alpha_{3,6} = \alpha_{4,7} = 1$ *and* $\beta_{5,3} = 1$. *Arcs* $(2,5)$ *and* $(3,6)$ *are critical* ($z_{2,5} = z_{3,6} = 1$), *milestone 8 is non-early* ($z_8 = 1$) *and milestone 9 is early* ($z_9 = 0$), *so* $F_k = \{(2,5),(3,6)\}, B_k = \emptyset, C_k = \{(5,3)\}$ *and* $D_k = \{9\}$. *The cut* $\omega_{inc}(X^*)$ *is profitable, since the saving from extending* $(2,5)$ *and* $(3,6)$ *(=60+50) outbalances the increased penalty related to milestone 8 (=100). So we generate the constraint*

$$x_{2,5} + x_{3,6} \leq 1 + z_{5,3} + z_9.$$

If arc $(5,3)$ *becomes critical (i.e.,* $z_{5,3}$ *turns to 1), we turn off the constraint, as the crashing cost* $c_{53} = 20$ *is subtracted from the saving, making therefore the cut no more profitable (*$60+50-100-20 < 0$*). Similarly, if milestone 9 becomes non-early, we turn off the constraint since subtracting* $q_{9,u} = 100$ *would make the cut no more profitable.*

7. Computational experiments

In this section, we describe the computational experiments performed to test the proposed approach based on lazy constraint generation, from now on denoted as ILP-LC. In Section 7.1 the experimental setup and settings are described, Section 7.2 discusses the algorithm performance and Section 7.3 deals with further experiments, designed to gain managerial insights on how milestone-related parameters may affect the project makespan.

7.1. Experimental settings and implementation details

A problem instance is primarily characterized by the number of activities n, the number of milestones $|N_m|$ and the number of agents a. The project networks used in the experiments were generated using RanGen1 [Demeulemeester et al. 2003] as activity-on-node networks, which were then converted to activity-on-arc networks using the algorithm described in [Demeulemeester and Herroelen 2002]. The *order strength* (i.e., the ratio between the number of precedence relations in the transitive closure of the project network and the theoretical maximum number of precedence relations in the network) was fixed to 0.3 because, as observed in [Kolisch et al. 1992], this is a typical value for realistic project networks. Such a relatively small value makes these instances challenging from the computational viewpoint (see [Herroelen and De Reyck 1999]). The activity durations generated by RanGen1 were used as *crash durations* $\underline{p}_{ij}$. The normal duration $\overline{p}_{ij}$ of each activity (i,j) is obtained by adding to the crash duration $\underline{p}_{ij}$ a random number in [1,20]. *Crashing costs* are uniformly distributed in [10,200].

The reward sharing policy w_u is fixed according to the so-called *available-cost sharing* rule (see [Briand et al. 2017]), i.e.,

$$w_u = \frac{\sum_{(i,j)\in E_u} c_{ij}(\overline{p}_{ij} - \underline{p}_{ij})}{\sum_{(i,j)\in E} c_{ij}(\overline{p}_{ij} - \underline{p}_{ij})},$$

which tends to give higher incentive to agents having high cost investments. The tightness of the milestones' due dates d_m is controlled by the parameter $\alpha \in [0,1]$ such that:

$$d_m = t_m^{min} + \alpha(t_m^{max} - t_m^{min}).$$

A problem instance is also characterized by rewards/penalties parameters, which were set up as follows:

- the incentive - attractiveness parameter δ is picked in the interval $\in [0,1]$, such that the daily reward $\pi = \delta W^{\max}$, where $W^{\max}$ corresponds to the price of the most expensive cut in the graph, i.e.:

$$W^{\max} = \max_{X\subseteq N}\{\sum_{(i,j)\in\omega^+(X)} c_{ij} - \sum_{(i,j)\in\omega^-(X)} c_{ij}\};$$

	with minimum number of arcs					optimal cuts only				
n	Tcpu [s]	Tcpu CB [s]	Call-back [%]	#LC [−]	#Call-back [−]	Tcpu [s]	Tcpu CB [s]	Call-back [%]	#LC [−]	#Call-back [−]
20	0.17	0.15	87.5	6.28	3.92	0.80	0.67	83.0	50.73	52.46
30	0.34	0.26	75.8	7.87	5.26	19.16	10.94	57.1	673.44	676.04

Table 2: The impact of profitable cut with the minimum number of arcs.

- penalties attached to milestones are controlled by parameter $\beta \in [0, 1]$ such that the per-day tardiness penalties q_{mu} related to milestone m and agent u are identical and equal $\beta\pi$, provided that at least one activity belonging to agent u precedes milestone m (otherwise q_{mu} is set to 0).

The ILP-LC approach has been implemented in C#, and runs on a personal computer with an Intel Core i7 processor at 2.7 GHz with 8 GB of RAM. The underlying MILP model was solved with Gurobi solver version 8.1.1.

Before we present individual experiments, let us discuss now how the ILP-LC solving approach has been implemented. We conducted a set of preliminary experiments on instances with $n = 20$ and $n = 30$ to compare the performance of generating cuts with the smallest number of arcs vs. generating the most profitable ones, as discussed in Section 6.2 (Problems (34)–(38) and (43)–(47)). Table 2 compares the average CPU time of both strategies (column "Tcpu"), the average time spent in the call-back function (column "Tcpu CB"), and the percentage of time spent in the call-back function (column "Call-back"). In addition, the table presents the average number of generated lazy constraints (column "#LC") and the average number of raised call-backs per instance (column "#Call-back"). Both data sets show that the strategy that uses optimal cuts generates many more lazy constraints, which harms CPU time. Therefore, the algorithm evaluated below uses only the strategy generating cuts with the smallest number of arcs.

7.2. Performance analysis

We first investigate the efficiency of our ILP-LC approach vs. the compact ILP proposed in [Briand et al. 2017] (ILP-COMPACT) to solve problem instances without milestones (hence $|N_m|, \alpha$ and β do not apply in this context). We consider the same benchmark as in [Briand et al. 2017] (available at `https://github.com/CTU-IIG/MAPSP`) with the following parameters: $n \in \{20, 40, 60, 80\}$, $a = 5$ and $\delta = \pi/W^{\max} = 0.4$. The benchmark set has 100 instances for each value of n.

Table 3 reports on the size of each ILP formulation in terms of the number of variables ($\#var$) and constraints ($\#constr$). The comparison shows that ILP-COMPACT uses more than 12 times more variables and more than 4 times more constraints. So, ILP-LC has significantly lower memory requirements than ILP-COMPACT. The table also illustrates the average number of generated lazy constraints and the average percentage of the CPU time spent in the call-back function. Value #LC shows that the average number of generated lazy constraints is pretty small. The same holds for the average number of call-back executions denoted #Call-back. The percentage of time spent in the call-back function (column "Call back") is pretty large for small instances, but it sharply decreases

	ILP-LC						ILP-COMPACT		
n	#var [-]	#constr [-]	#LC [-]	#Call-back [-]	Tcpu [s]	Call-back [%]	#var [-]	#constr [-]	Tcpu [s]
20	323.7	555.3	6.3	3.9	0.2	88	4134.1	2542.8	0.05
40	1021.6	1785.4	10.2	6.9	0.73	59	13018.6	7987.5	0.74
60	2040.9	3600.5	15.3	10.2	2.71	39	26064.6	15947.4	5.09
80	3336.3	5920.9	31.7	20.9	18.1	19	42696.2	26069.5	42.18

Table 3: Comparison between ILP-LC and ILP-COMPACT (different number of activities).

	ILP-LC						ILP-COMPACT		
a	#var [-]	#constr [-]	#LC [-]	#Call-back [-]	Tcpu [s]	Call-back [%]	#var [-]	#constr [-]	Tcpu [s]
2	1021,6	1785,4	10.2	11.2	0.6	50	4420.4	7469.7	9.5
4	1021,6	1785.4	10.3	7.7	0.7	56	6794.4	11149.0	6.8
8	1021,6	1785,4	9.1	4.4	0.6	67	11542.6	18507.6	5.4
16	1021,6	1785,4	12.4	3.3	0.7	87	21038.9	33224.7	3.8

Table 4: Comparison between ILP-LC and ILP-COMPACT (different number of agents).

as n grows. A comparison of their CPU times is also provided in Table 3. On small-size instances (20 activities), one can observe that ILP-LC is slower (see $Tcpu$), which is mainly due to the time overhead associated with the call and execution of the call-back function. On larger instances, ILP-LC is faster and on the largest instances it is more than twice faster than ILP-COMPACT.

The influence of the number of the agents is studied in Table 4 assuming instances with $n = 40$. Columns $\#var$ and $\#constr$ show that the size of the ILP formulation is independent of the number of agents in the case of ILP-LC while the size of ILP-COMPACT significantly grows with a. ILP-LC is always less memory-demanding and faster than ILP-COMPACT. Note that the CPU time of ILP-LC is practically constant while that of ILP-COMPACT decreases with a.

The above results suggest that, when milestones are considered, ILP-LC is a more suitable approach than ILP-COMPACT. Moreover, the formulation approach of ILP-COMPACT uses duality relations (max-flow/min-cut and min-flow/max-cut), which may not be easily extended to the case with milestones (see Section 6.3).

A second set of experiments has been made to assess the efficiency of ILP-LC on instances with milestones. We chose $a \in \{3, 4, 5\}$, $n \in \{20, 40, 60\}$, $\delta = 0.2$, $|N_m| = 5$, $\alpha = 0.5$ and $\beta = 0.04$. Parameters α, β, and δ are assumed fixed in this experimental set, as their impact is studied in the next section.

Tables 5 and 6 analyze how the number of activities and agents respectively affect the algorithm performance. Results in Table 5 assume $a = 4$ while in Table 6 we assume $n = 40$. We first observe that the algorithm is able to solve instances with 60 activities in a reasonable amount of time.

n [−]	Tcpu [s]	Tcpu CB [s]	Call-back [%]	#LC [−]	#Call-back [−]
20	0.7	0.6	89.7	39.8	19.0
40	15.0	10.0	66.5	505.2	181.3
60	12029.6	297.1	2.5	11091.8	3240.9
40*	19.4	15.9	82.1	319.7	322.9
60*	2611.6	377.8	14.5	4879.3	4885.6

Table 5: CPU performance for different problem sizes (number of activities).

a [−]	Tcpu [s]	Tcpu CB [s]	Call back [%]	#LC [−]	#Call-back [−]
3	27.0	15.4	57.1	915.2	374.9
4	15.0	10.0	66.5	505.2	181.3
5	10.4	7.6	72.9	344.9	108.9

Table 6: CPU performance for different problem sizes (number of agents).

Instances with fewer agents are generally computationally more demanding. The ILP-LC algorithm generates more lazy constraints (column #LC) compared to the previous experiment without milestones, however, for the largest instances ($n = 60$) the time spent in the call-back function (column "Tcpu CB") is only 2.5% of the overall CPU time (column "CPU time"). The higher frequency of lazy constraints is caused by the penalization for tardy milestones that increases the chance that a solution is not a Nash equilibrium. The rows in Tables 5 with "*" in the first column refer to the strategy in which we generate only one lazy-constraint for every solution which in not Nash. When $n = 40$, this strategy is definitely worse than adding all lazy-constraints (see Algorithm 1). On the other hand, for $n = 60$, the situation is the opposite. In this case, the CPU time of the strategy adding all lazy constraints is much worse. The algorithm adds 11091 lazy constraints on average, which significantly degrades the performance of the algorithm.

Table 6 shows that the number of generated lazy constraints decreases as the number of agents grows, and thus also the overall CPU time decreases. In fact, for a given n, there are typically more profitable cuts to be considered in the lazy constraint generation mechanism in an instance with fewer agents, although the difference is not very large. On the other hand, the percentage of CPU time spent by the call-back function increases with increasing a, since the call-back finds more profitable cuts, i.e., for each agent one increasing and one decreasing cut.

7.3. Impact of milestones

In this section we address another set of experiments aimed at highlighting how milestone-related parameters affect the project makespan. Such parameters are (i) the tightness of milestone due dates (expressed by parameter α), (ii) the number of milestones $|N_m|$, and (iii) the value of tardiness

penalties (expressed by parameter β). In this set of experiments, we fix $n = 40$ and $a = 4$, so that the computational effort is not large (all the instances of this experiment were solved within less than twelve seconds on average). Then, we consider the values $\delta \in \{0.1, 0.2, 0.4\}$, and for each δ the following combinations of parameters: $\alpha \in \{0.25, \underline{0.5}, 0.75\}$, $|N_m| \in \{1, 3, \underline{5}\}$ and $\beta \in \{0.008, \underline{0.04}, 0.072\}$. Underlined values are the *nominal* values, i.e., when analyzing the impact of one parameter in $\{\alpha, |N_m|, \beta, \}$, the other two are kept fixed to their nominal value. Note that $|N_m| = 1$ means that the only milestone is project completion (no intermediate milestones). For each scenario, 100 new instances were generated and solved.

Tables 7, 8, and 9 report on the influence of $\alpha, |N_m|$ and β, respectively, on the average makespan D for three different values of daily reward δ. The makespan is expressed as a percentage of the size of the interval $[\underline{D}, \overline{D}]$, and it is denoted by $D^{rel}(S^*)$:

$$D^{rel}(S^*) = 100 \cdot \frac{D(S^*) - \underline{D}}{\overline{D} - \underline{D}},$$

where S^* is the optimal strategy for MAPSM. We also report on total tardiness, denoted as T, and penalty $\sum_{A_u \in \mathcal{A}} \sum_{m \in N_m} q_{mu} T_m$.

All three tables show that the presence of milestones has a significant impact on the makespan when the daily reward is low, i.e., $\delta = 0.1$. In such a scenario, a significant reduction of the makespan is attained when due dates are tighter ($D^{rel}(S^*)$ decreases by 18,6% when reducing α from 0.75 to 0.25), or when per-day tardiness penalty is higher ($D^{rel}(S^*)$ decreases by 12,6% when increasing β from 0.008 to 0.072). Increasing the number of milestones from 1 to 5 only reduces $D^{rel}(S^*)$ by less than 5%. So, setting tighter milestone due dates and higher tardiness penalties appear as sensible strategies to achieve a lower makespan, rather than setting a large number of milestones. However, the effectiveness of these strategies changes if the daily reward π grows. When $\delta = 0.2$ (hence, twice than in the previous scenario), tightening the due dates still results in a significant makespan decrease (from 38.2% to 28.8%), while the value of tardiness penalties becomes less significant. Finally, when $\delta = 0.4$, the daily reward π appears large enough to make project makespan largely insensitive to the other three parameters.

Total tardiness and penalty are indicators that need to be taken into account in the analysis of the results. For instance, the scenario with $\alpha = 0.25$ and $\delta = 0.1$ in Table 7 indicates that tightening the due dates significantly decreases the project makespan; however, it also leads to high penalties that may not be acceptable for the agents. In comparison, $\alpha = 0.5$ is a more realistic scenario, which is only slightly worse in terms of project makespan. Another interesting case is $\beta = 0.008$ and $\delta = 0.1$ in Table 9. Unlike the previous example, the penalty is not very high, and the scenario could be acceptable for agents. Nevertheless, this case leads to very high total tardiness as agents have low incentive to decrease the duration of their activities to satisfy milestones, which may induce many conflicts between the project owner and the agents.

This experiment shows some interesting insights for the project manager. In order to expedite the project, the project owner should mainly find the best combination of appropriate due date penalties

and daily rewards. Indeed, even if β is chosen as a relatively small fraction of the daily reward, setting suitable milestones may significantly impact project makespan. On the other hand, milestone parameters need to be appropriately set in order to achieve the goal of makespan reduction while keeping tardiness penalties acceptable for the agents. In this respect, our model can be used as a suitable tool to anticipate the effects that a particular milestone setting policy may have on the overall project performance.

δ [−]	α [−]	$D^{rel}(S^*)$ [%]	T [−]	penalty [−]	CPU time [s]	#LC [−]	#Call-back [−]
0.4	0.25	11.9	1.04	132.8	7.3	266.1	109.2
	0.5	13.3	0.38	38.1	6.6	223.1	94.8
	0.75	13.8	0.12	9.6	6.2	198.2	85.3
0.2	0.25	28.8	2.80	424.8	11.0	618.7	218.9
	0.5	36.0	1.17	150.1	9.0	505.2	181.3
	0.75	38.2	0.45	51.5	7.5	421.2	152.8
0.1	0.25	43.5	5.80	961.2	11.7	660.2	219.7
	0.5	50.8	1.94	286.2	8.8	540.4	176.6
	0.75	62.1	0.79	106.5	6.9	424.4	139.2

Table 7: Results for different values of due date tightness (α).

δ [−]	$\lvert N_m\rvert$ [−]	$D^{rel}(S^*)$ [%]	T [−]	penalty [−]	CPU time [s]	#LC [−]	#Call-back [−]
0.4	5	13.3	0.38	38.1	4.7	223.1	94.8
	3	13.8	0.30	13.5	3.6	161.1	76.1
	1	14.0	0.00	0.0	3.1	122.6	69.1
0.2	5	36.0	1.17	150.1	9.1	505.2	181.3
	3	37.0	0.93	59.8	5.7	305.8	121.6
	1	38.0	0.01	0.3	4.1	204.3	93.8
0.1	5	50.8	1.94	286.2	8.8	540.4	176.6
	3	53.2	2.23	188.5	4.7	283.8	101.2
	1	55.1	2.89	99.7	3.2	183.1	74.6

Table 8: Results for different numbers of milestones ($|N_m|$).

8. Conclusions

In this paper, we presented a model for finding equilibria in a project scheduling environment in which activities belong to different agents. In order to expedite the process, the project owner may set rewards for early project completion, as well as penalties if some intermediate milestones are

δ [−]	β [−]	$D^{rel}(S^*)$ [%]	T [−]	penalty [−]	CPU time [s]	#LC [−]	#Call-back [−]
	0.008	13.8	0.60	13.2	4.9	223.7	100.3
0.4	0.04	13.3	0.38	38.1	4.6	223.1	94.8
	0.072	12.7	0.24	43.6	4.0	198.1	82.9
	0.008	38.0	1.94	51.6	9.0	501.6	183.6
0.2	0.04	36.0	1.17	150.1	9.1	505.2	181.3
	0.072	35.0	0.67	147.7	8.4	482.8	168.8
	0.008	61.0	5.45	167.5	8.5	523.5	173.6
0.1	0.04	50.8	1.94	286.2	8.8	540.4	176.6
	0.072	48.4	0.88	217.2	7.9	487.4	157.0

Table 9: Results for different values of tardiness penalty charges (β).

not respected. The project owner wants to achieve a stable solution having the minimum makespan. Here, stability refers to agents having no interest in changing the duration of their activities, and it is captured by the concept of Nash equilibrium. We developed a lazy constraint-based solution scheme, where the fact of whether or not a solution is a Nash equilibrium is checked in a call-back function of an ILP solver. The power of the algorithm lies in the efficient generation of lazy constraints at the run time of the algorithm. If a solution is not a Nash equilibrium, the algorithm rules out not only the particular solution but also all solutions having a particular cut in the residual graph. Thus, more solutions that are not Nash equilibria are cut off at once. We further improve the performance of the algorithm by searching cuts with the minimum number of arcs and, in this way, prune the solution space even more, as it is illustrated in the experimental results.

Using the algorithm, one can assess the effect of specific parameter configurations (incentives, penalties, due date tightness) on makespan performance. Our experiments show that while per-day rewards are the most effective lever for reducing project makespan, setting a small number of intermediate milestones may be a partial substitute when the budget for rewards is low.

While we established some results for the computation of the minimum-makespan Nash equlibrium, the quality of such equilibrium with respect to possibly non-Nash solutions might be investigated. This issue has been addressed in [Agnetis et al.2019] for the model without milestones. The makespan values of the best and worst Nash equilibria were compared against the best makespan of a *base schedule*, characterized by the only (weak) requirement that no agent has negative revenue. It turned out that the price of anarchy (% increase of makespan in the worst equilibrium w.r.t. the best makespan of a base schedule) does not significantly depend on the number of agents, while it is significantly affected by the average range of activity durations (the price of anarchy grows linearly with it). As the computation of the worst Nash equilibrium is also very complex, we think that performing a similar analysis for the more general model considered here would be out of the scope of the present paper.

Future research might also address other issues:

- *Variable milestone setting.* In this research, we assumed that milestones and their parameters are given, being set up by the project owner. Using the proposed model, the project owner can compute several scenarios for different milestone settings (as in the experiments in Section 7.3) and, based on that, propose one selected scenario to the agents (sub-contractors). In fact, a different approach could be to define milestones and milestone-related parameters as variables, i.e., let the model place the milestones in the project network and set due dates and penalties so that the project makespan is minimized while having a Nash equilibrium. However, such a comprehensive model would require developing a different algorithmic approach and it might turn out to be considerably more complex to solve.

- *Cash flow modeling.* In this paper, cash flows are supposed to take place simultaneously (e.g., at the end of the project), while net present value-related objectives might be pursued by both the agents and the project owner.

- *Heuristics.* In view of the inherent complexity of the problem, one important venue for future research is to devise algorithms to efficiently find good-quality Nash equilibria.

- *Mechanism design issues.* In this paper we have considered that agents choose their strategies *independently* and *simultaneously*, as it is assumed in noncooperative games. If this is not the case, and the agents sequentially declare their respective strategies, an agent's benefit can largely depend on the sequence chosen. This might in turn configure a cooperative sequencing game, in which agents might trade their position in the sequence with utility transfers.

- *Activity assignment.* In this paper we addressed the situation in which the activities are preassigned to the agents, and each agent acts individually. A different situation is when the project owner decides, up to a certain extent, how to assign activities to the agents, and some activity might even be jointly carried out by more than one agent. This would require developing different models, possibly drawing from cooperative game theory.

9. Acknowledgements

This work was supported by the European Regional Development Fund under the project AI&Reasoning (Reg. No. CZ.02.1.01/0.0/0.0/15_003/0000466).

References

[Agnetis et al., 2015] Agnetis, A., Briand, C., Billaut, J.-C., Šůcha, P., Nash equilibria for the multi-agent project scheduling problem with controllable processing times, *Journal of Scheduling*, 18(1), 15–27, 2015.

[Agnetis et al., 2019] Agnetis, A., Briand, C., Ngueveu, S.U., Šůcha, P., Price of anarchy and price of stability in multi-agent project scheduling, *Annals of Operations Research*, 285, 97—119, 2020.

[Ahuja et al. 1993] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network Flows*. Prentice-Hall, Upper Saddle River, NJ, USA.

[Anshelevich et al. 2008] Anshelevich E., Dasgupta A., Kleinberg J., Tardos É., Wexler T., Roughgarden T., The price of stability for network design with fair cost allocation, *SIAM Journal on Computing*, 38(4), 1602—1623, 2008.

[Arık et al. 2019] Arık O.A., Kose, E., Forrest J.Y.-L., Project Staff Scheduling with Theory of Coalition, Group Decision and Negotiation, 28, 827–847, 2019.

[Averbakh 2010] Averbakh, I., Nash equilibria in competitive project scheduling, *European Journal of Operational Research*, 205(3), 552—556, 2010.

[Bergantiños and Lorenzo 2019] Bergantiños, G., Lorenzo, L., How to apply penalties to avoid delays in projects, *European Journal of Operational Research* 275, 608–620, 2019.

[Brânzei et al 2002] Branzei, R., Ferrari, G., Fragnelli, V., Tijs, S., Two Approaches to the Problem of Sharing Delay Costs in Joint Projects, *Annals of Operations Research*, 109, 359–374, 2002.

[Briand et al. 2017] Briand, C., Ngueveu, S.U., Šůcha, P., Finding an optimal Nash equilibrium to the multi-agent project scheduling problem, *Journal of Scheduling*, 20, 475–491, 2017.

[Brucker et al 1999] Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E., Resource-constrained project scheduling: Notation, classification, models, and methods, *European Journal of Operational Research*, 112, 3–41, 1999.

[Christodoulou and Koutsoupias 2005] Christodoulou, G. and E. Koutsoupias, On the Price of Anarchy and Stability of Correlated Equilibria of Linear Congestion Games, in Brodal G.S., Leonardi S. (eds), Algorithms – ESA 2005. Lecture Notes in Computer Science, vol 3669, 59–70, Springer, Berlin, Heidelberg, 2005.

[Ciurea and Ciupalâ 2004] Ciurea, E. and Ciupalâ, L. (2004). Sequential and parallel algorithms for minimum flows. *Journal of Applied Mathematics and Computing*, 15:53–75.

[Confessore et al. 2007] Confessore, G., Giordani, S., and Rismondo, S., A market-based multi-agent system model for decentralized multi-project scheduling, *Annals of Operations Research*, 150, 115–135, 2007.

[Dantzig et al. 1959] Dantzig, G. B. and Fulkerson, D. R. and Johnson, S. M., On a Linear-Programming, Combinatorial Approach to the Traveling-Salesman Problem, *Operations Research*, Vol.7, No. 1, 58–66, 1959.

[De Ita Luna et al. 2015] De Ita Luna, G., Zacarias-Flores, F., Altamirano-Robles, L.C., Finding Pure Nash Equilibrium for the Resource-Constrained Project Scheduling Problem, *Computación y Sistemas*, Vol. 19, No. 1, 17-–27, 2015.

[De Reyck et al. 1999] De Reyck, B., Herroelen, W., The multi-mode resource-constrained project scheduling problem with generalized precedence relations, *European Journal of Operational Research* 119, 538–556, 1999.

[Demeulemeester and Herroelen 2002] Demeulemeester, E. L. and Herroelen, W. S., *Project Scheduling - A Research Handbook*. Springer Science and Business Media, 2006.

[Demeulemeester et al. 2003] Demeulemeester, E., Vanhoucke,M., Herroelen,W., Rangen: A random network generator for activity-on-the-node networks, *Journal of Scheduling*, 6, 17—38, 2003.

[Estévez-Fernández 2012] Estévez-Fernández, A., A game theoretical approach to sharing penalties and rewards in projects, *European Journal of Operational Research*, 216(3), 647–657, 2012.

[Herroelen and De Reyck 1999] Herroelen, W.S. and De Reyck, B., Phase transitions in project scheduling. *Journal of the Operational Research Society*, 50(2), 148—156, 1999.

[Kolisch et al. 1992] Kolisch, R., Sprecher, A., Drexl, A., Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems, Institut für Betriebswirtschaftslehre Christian-Albrechts-Universität zu Kiel, working paper no.301, 1992.

[Kress et al. 2018] Kress, D., Meiswinkel, S. Pesch, E., Mechanism design for machine scheduling problems: classification and literature overview, *OR Spectrum*, 40, 583—611, 2018.

[Moradi et al. 2019] Moradi, M., Hafezalkotob, A. and Ghezavati, V., Sustainability risk management in a cooperative environment under uncertainty, *Kybernetes*, Vol. 48 No. 3, pp. 385-406, 2019.

[Phillips and Dessouky 1977] Phillips, S. and Dessouky, M. I., Solving the project time/cost tradeoff problem using the minimal cut concept, *Management Science*, 24(4), 393–400, 1977.

[Spalek 2005] Spalek, S., Critical success factors in project management. To fail or not to fail, that is the question! presented at PMI Global Congress 2005–EMEA, Edinburgh, Scotland. Newtown Square, PA: Project Management Institute, 2005.

[Van Eynde 2017] Van Eynde, R., Multi-project scheduling - The application of a decoupled schedule generation scheme and a game mechanic, Master's Dissertation in Business Engineering, Universiteit Gent, 2017.

[Varakantham and Fu 2017] Varakantham, P., Fu, N., Mechanism Design for Strategic Project Scheduling, Research Collection School Of Information Systems, Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, 4433–4439, 2017.

Appendix

A.1 Finding any Nash equilibrium

The optimal solution to the following linear program defines a strategy S_L which is a Nash equilibrium.

$$
\begin{aligned}
\max \quad & \textstyle\sum_{u\in\mathcal{A}} \alpha_u(w_u(\bar{D}-t_r) - \sum_{\forall(i,j)\in E_u} c_{ij}(\overline{p}_{ij}-p_{ij}) - \sum_{\forall m\in N_m} q_{mu}T_m) \\
s.t. \quad & \\
& t_j - t_i - p_{ij} - s_{ij} = 0 && \forall (i,j)\in E \\
& \underline{p}_{ij} \le p_{ij} \le \overline{p}_{ij} && \forall (i,j)\in E \\
& t_0 = 0 && \\
& t_m - d_m \le T_m && \forall m \in N_m \\
& t_i, T_m, p_{ij} \in \mathbb{Z}_{\ge 0},\ s_{ij}\in\mathbb{R}_{\ge 0} &&
\end{aligned}
$$

where the weights $\alpha_1 >> \alpha_2 >> \cdots >> \alpha_a$ are such that lexicographic optimality is guaranteed.

A.2 Proof of Theorem 2

Proof. Consider a decreasing cut $\omega^{(u)}_{dec}(X)$ such that $W^{(u)}_{dec}(X) > \tilde{W}^{(u)}_{dec}(X)$, and let $\tilde{M}$ be the set of milestones for which a multiple makespan reduction is achieved. From the above discussion, this means that there is a set $\tilde{U}$ of nodes ($\tilde{U}\subseteq X$), which are not reachable from the initial node through a path fully contained in X. Let us, therefore, consider the node subset $\tilde{X}$ obtained removing from X all the nodes of $\tilde{U}$, and consider a new cut $\omega^{(u)}_{dec}(\tilde{X}) = \omega^{(u)}_{dec}(X\setminus\tilde{U})$. Now, in $\omega^{+(u)}_{dec}(\tilde{X})$ there can be no arc (k,i) such that $i\in\tilde{U}$, since otherwise in X there would have been a path from 0 to a node of $\tilde{U}$, and the makespan of some milestone in $\tilde{M}$ would not have been multiply reduced by applying $\omega^{(u)}_{dec}(X)$. As a consequence, considering the new cut $\omega^{(u)}_{dec}(\tilde{X})$, one has that $\omega^{+(u)}_{dec}(\tilde{X}) \subseteq \omega^{+(u)}_{dec}(X)$ (i.e., the set of forward arcs has not been enlarged), and applying the cut $\omega^{(u)}_{dec}(\tilde{X})$ no milestone in $N\setminus\tilde{X}$ experiences multiple reductions. So now the set $N\setminus\tilde{X}$ includes exactly all the milestones affected by the cut, and in conclusion, $\tilde{W}^{(u)}_{dec}(\tilde{X}) = W^{(u)}_{dec}(\tilde{X}) < W^{(u)}_{dec}(X)$. Hence, given any cut $\omega^{(u)}_{dec}(X)$, even if $\tilde{W}^{(u)}_{dec}(X) < W^{(u)}_{dec}(X)$, there is certainly another cut $\omega^{(u)}_{dec}(\tilde{X})$ such that $\tilde{W}^{(u)}_{dec}(\tilde{X}) = W^{(u)}_{dec}(\tilde{X}) < W^{(u)}_{dec}(X)$, so indeed minimizing $W^{(u)}_{dec}(X)$ is equivalent to minimizing $\tilde{W}^{(u)}_{dec}(X)$. □

A.3 Proof of Theorem 3

Proof. Suppose that there is an increasing cut $\omega^{(u)}_{inc}(X)$ such that $W^{(u)}_{inc}(X) < \tilde{W}^{(u)}_{inc}(X)$. This means that there is a set $\tilde{M}$ of milestones such that, even if they lay in $N\setminus X$, their respective makespans are not increased by applying the cut. Let us, therefore, consider the node subset $\tilde{N}\subset N\setminus X$ including all the nodes of $N\setminus X$ that are on some critical path leading to some node of $\tilde{M}$, and consider a new cut $\omega^{(u)}_{inc}(\tilde{X}) = \omega^{(u)}_{inc}(X\cup\tilde{N})$. Now, observe that in $\omega^{-(u)}_{inc}(\tilde{X})$ there can be no arc (k,j) such that $j\in\tilde{N}$,

since otherwise the makespan of some milestone in $\tilde{M}$ would have been affected by the increasing cut $\omega_{inc}^{(u)}(X)$. As a consequence, $\omega_{inc}^{+(u)}(\tilde{X}) \supseteq \omega_{inc}^{+(u)}(X)$ (i.e., passing from $\omega_{inc}^{(u)}(X)$ to $\omega_{inc}^{(u)}(\tilde{X})$, the set of forward arcs may have been enlarged), but the set of affected milestones in $N \setminus \tilde{X}$ is smaller than $N \setminus X$. The set $N \setminus \tilde{X}$ includes exactly all the milestones affected by the cut, so in conclusion, $\tilde{W}_{inc}^{(u)}(\tilde{X}) = W_{inc}^{(u)}(\tilde{X}) > W_{inc}^{(u)}(X)$. Hence, given any cut $\omega_{inc}^{(u)}(X)$, if $\tilde{W}_{inc}^{(u)}(X) > W_{inc}^{(u)}(X)$, there is certainly another cut $\omega_{inc}^{(u)}(\tilde{X})$ such that $\tilde{W}_{inc}^{(u)}(\tilde{X}) = W_{inc}^{(u)}(\tilde{X}) > W_{inc}^{(u)}(X)$, so indeed maximizing $W_{inc}^{(u)}(X)$ is equivalent to maximizing $\tilde{W}_{inc}^{(u)}(X)$. □