A Non-cooperative Game-Theoretic Approach for Conflict Resolution in Multi-agent Planning

Jordán, Jaume; Torreño, Alejandro; de Weerdt, Mathijs; Onaindia, Eva

DOI
10.1007/s10726-020-09703-0

Publication date
2020

Document Version
Final published version

Published in
Group Decision and Negotiation

Citation (APA)
Jordán, J., Torreño, A., de Weerdt, M., & Onaindia, E. (2020). A Non-cooperative Game-Theoretic Approach for Conflict Resolution in Multi-agent Planning. *Group Decision and Negotiation*, *30*(1), 7-41. https://doi.org/10.1007/s10726-020-09703-0

Group Decision and Negotiation
https://doi.org/10.1007/s10726-020-09703-0

A Non-cooperative Game-Theoretic Approach for Conflict Resolution in Multi-agent Planning

Jaume Jordán[1] · Alejandro Torreño[1] · Mathijs de Weerdt[2] · Eva Onaindia[1]

Abstract
This paper presents FENOCOP, a game-theoretic approach for solving non-cooperative planning problems that involve a set of self-interested agents. Each agent wants to execute its own plan in a shared environment but the plans may be rendered infeasible by the appearance of potential conflicts; agents are willing to coordinate their plans in order to avoid conflicts during a joint execution. In order to attain a conflict-free combination of plans, agents must postpone the execution of some of their actions, which negatively affects their individual utilities. FENOCOP is a two-level game approach: the General Game selects a Nash equilibrium among several combinations of plans, and the Scheduling Game generates, for a combination of plans, an executable outcome by introducing delays in the agents' plans. For the Scheduling Game, we developed two algorithms that return a Pareto optimal and fair equilibrium from which no agent would be willing to deviate.

Keywords Planning · Multi-agent planning · Game theory · Nash equilibrium · Pareto optimal · Fairness

✉ Jaume Jordán
jjordan@dsic.upv.es

Alejandro Torreño
atorreno@dsic.upv.es

Mathijs de Weerdt
m.m.deweerdt@tudelft.nl

Eva Onaindia
onaindia@dsic.upv.es

[1] Institut Valencià d'Investigació en Intel·ligència Artificial (VRAIN), Universitat Politècnica de València, Camino de Vera, s/n, 46022 Valencia, Spain

[2] EEMCS, Algorithmics, Delft University of Technology, P.O. Box 5031, 2600 GA Delft, The Netherlands

Published online: 05 September 2020

1 Introduction

Planning is the ability of selecting the appropriate action, among a set of alternatives, in a particular situation considering the conditions of the world state, the knowledge of the environment, the impact of the effects of the action, and the extent to which the decision helps in finding a solution for the planning task. The single-agent planning problem is conceived as a search process by which a single entity synthesizes a set of actions (plan) to reach a goal from an initial situation (Ghallab et al. 2004). There exists a large variety of planners, many of which have participated in the different editions of the International Planning Competition (IPC)[1] event held to date. While the first IPCs aimed at evaluating the performance of solvers in deterministic planning, the most recent editions have extended the competition to temporal planning and probabilistic planning.

Planning with multiple agents involves coordinating planning, control and execution activities. In general, Multi-Agent Planning (MAP) deals with the problem of automated planning in domains where multiple agents plan and act together in a shared environment (Weerdt and Clement 2009; Nguyen and Katarzyniak 2009). Motivated by the growing research in MAP, the first Competition of Distributed and Multi-Agent Planners (CoDMAP) was held in 2015 in the context of the IPC (Komenda et al. 2016).

MAP is characterized by two main factors, the type of task that agents are aimed to solve and the type of environment. In this work, we restrict our attention to deterministic environments so we assume that the only interactions that can emerge are due to the coordination of the agents' activities and not due to exogenous events or uncertainty in the world. On the other hand, the typology of the MAP task alongside the nature of the agents, altruistic/cooperative versus self-interested/strategic agents, determine the characteristics of a MAP approach.

In a MAP context, a task is defined as a set of common goals to be jointly achieved by several agents or as a number of goal sets in which each set must be independently achieved by a single agent. When agents work together to achieve a common goal or help each other to achieve their goals, this is called cooperative planning. Cooperative planning has gained much attention lately within the planning community to promote the resolution of automated planning problems among multiple altruistic agents (Torreño et al. 2014; Komenda et al. 2016). In this case, agents have no private interests and they work for the common benefit of the group, either solving a common MAP task or a set of independent tasks (Torreño et al. 2018). However, in the case of self-interested agents that have their own strategies, agents are designed to make their strategic behavior prevail over the others when solving the particular planning task.

The mainstream in MAP with self-interested planning agents is handling situations which involve interactive decision making with possibly divergent (conflicting) interests. Game theory, the study of mathematical models of conflict and cooperation

[1] http://www.icaps-conference.org/index.php/Main/Competitions.

between rational and self-interested agents (Osborne and Rubinstein 1994; Von Neumann and Morgenstern 2007; Myerson 2013), arises naturally as a paradigm to address human conflict and cooperation within a competitive situation. Particularly, *non-cooperative game theory* (NCGT) is concerned with strategic equilibrium and individual utility maximization given the actions of the other actors of the game. Within NCGT, the purpose of an agent in the so-called strictly competitive games or zero-sum games is to find a strategy that is good for such agent and bad for the opponents. However, non-cooperative games that take place in non-strictly competitive settings feature agents with conflicting and complementary interests. Conflicts emerge as a consequence of all agents attempting to impose their strategic behavior in a common environment. But agents are also willing to cooperate by resigning their max-utility strategy for the sake of a joint solution that accommodates a strategy for all the participants. This type of games, which are modeled as non-zero-sum games or general-sum games, yield win–win strategies in which all participants can profit from the game one way or the other (Gillies 1959; Shoham and Leyton-Brown 2009).

Non-cooperative games have been applied in strictly-competitive MAP settings, also known as adversarial planning (Jensen et al. 2001; Sailer et al. 2007). The application of NCGT to non-strictly competitive MAP gives rise to the field commonly referred to as *non-cooperative MAP*. Unlike coalitional planning, which draws upon cooperative games where agents contract each other's behavior to form coalitions (Brafman et al. 2009; Crosby and Rovatsos 2011), in non-cooperative settings planning agents act independently to each other. Non-cooperative MAP has been used to solve a number of different tasks, which can be identified as a set of strategic agents aimed at solving a common MAP task or several individual planning tasks.

Common MAP Task When agents are designed to solve one single task, the strategic behavior lies in the different utilities or rewards that agents earn accordingly to the part of the task they solve. Solving a MAP task with self-interested agents requires either distributing the task (goal allocation) among the agents or calculating for each agent a plan that complies with its strategic behavior while optimizing some global criterion.

One common problem in non-cooperative MAP is to determine the goals to be achieved by each agent while maximizing their utility. Agents can exchange goals and subgoals through an auction, using their own heuristics or utility functions to determine when to auction and what to bid (Van Der Krogt and De Weerdt 2005). In cost-optimal planning problems, on the other hand, it is important to ensure that self-interested agents report truthfully their private information about its abilities or its cost. The work in Nissim and Brafman (2013) presents a mechanism design approach that optimizes social welfare where agents receive a payment that reflects the impact each agent's plan participation has on the other agents' plans. In general, solving a common task by a number of non-cooperative agents requires a strategy that combines the private interests of the agents with the global purpose of the task.

Multiple Individual Planning Tasks In this case, the problem consists of a series of planning tasks that have to be solved by self-interested agents that wish to plan autonomously. Agents are assumed to independently work on their own part of the planning problem either because of the existence of competitive

relationships or because the impossibility of communicating during the planning process. One first attempt in this direction is presented in Buzing et al. (2006), where agents not only plan autonomously but also do not want to revise their individual plans when the joint plan has to be assembled from the individual plans. Hence, the proposal in Buzing et al. (2006) is a pre-planning coordination method that ensures a feasible global solution to the multi-agent planning problem—whatever plans are chosen by the individual agents—by imposing (a minimal set of) additional constraints to the original planning problem.

Most commonly, agents apply a post-planning method by which they evaluate the interaction of their plans with the plans of the other agents and come up with a joint assembled plan. Game-theoretic approaches have been proposed to evaluate combinations of agents' plans and formalize equilibria solutions (Bowling et al. 2003). Other works define a classification hierarchy of plan combinations according to the level of goal satisfaction that each combination reports to the agents (Larbi et al. 2007). In this latter work, authors also define the game-theoretic notions to characterize joint plans that are Nash equilibrium (NE), a more widely accepted concept of solution that takes into account all agents' opportunities. While these two works mainly focus on introducing a formal definition of equilibria for MAP, a more recent proposal presents a general non-cooperative MAP framework where agents dispose of a number of plans and each plan reports a benefit to the agent accordingly to the number of goals achieved, duration of the plan (makespan) or cost of the actions (Jordán and Onaindía 2015). This two-game framework evaluates all possible schedules of the agents' plans so that the set of strategies of all the agents constitute a NE solution.

A Nash equilibrium joint plan is a stable solution in the sense that no agent has anything to gain by changing their plans. Nevertheless, further criteria are applicable to NE solutions without losing the stability provided by the equilibrium. The Pareto Optimality (PO) criterion helps remove those NE solutions which are improved by other NE solutions without decreasing the utility of any agent. Yet, albeit PO determines when the allocation of agents' schedules is optimal, it makes no statement about equality, thus possibly resulting in agents unsatisfied regarding their individual utilities. This limitation can be overcome by subsequently selecting *fair solutions* so as to balance out the individual satisfaction of the agents. The concept of *fairness* has been widely studied in voting theory, analyzing the compliance of voting methods with fairness criteria such as Majority, Condorcet or Monotonicity criteria (Brandt et al. 2016). The *maximin* principle of distributive justice, whereby a solution A is more fair than a solution B if the worst off agent in A is better off than the worst off agent in B, is often interpreted as a highly egalitarian principle; other works pinpoint that the fairest solution is achieved by maximizing the minimum utility subject to the envy-freeness constraint (Gal et al. 2018).

Albeit there have been a few attempts to establish the theoretical foundations of equilibrium solutions for non-cooperative MAP, no practical implementation exists to date. Following the line of the two-game proposal of Jordán and Onaindía (2015), we present here an enhanced version that contributes with the following features:

- A general framework, called FENOCOP, that solves non-cooperative MAP tasks for independent agents that plan autonomously; agents calculate a set of individual plans that solve their respective problems, and then engage in a game to select a plan schedule that allows them to execute their plans simultaneously in a common environment.
- Two novel game algorithms that allow agents to consistently synchronize the execution of their plans. Both algorithms find solutions compliant with *Pareto optimality* and *fairness*, thus balancing out the individual satisfaction of the agents. In this work, we opt for applying the egalitarian principle to return fair schedules, in a transparent way without need of requesting agents their preferences for the schedules.
- Empirical evaluation of the FENOCOP framework with a special emphasis on the performance of the two game algorithms. This is a relevant aspect of our contribution since game-theoretic approaches are rarely empirically tested.

The contents of the paper are organized as follows. Section 2 introduces the formal notions related to the planning task of the agents. Section 3 presents an overview of FENOCOP. Sections 4 and 5 outline the characteristics of the two-level game approach; the top-level General Game and the internal Scheduling Game. Section 6 is devoted to explain the two algorithms for solving the Scheduling Game, and Sect. 7 presents the empirical evaluation. Finally, Sect. 8 concludes and discusses the limitations of the model.

2 Planning Scenario

The problem we want to solve involves a set of n rational and self-interested agents $\mathcal{AG} = \{1, \ldots, n\}$, where each agent $i \in \mathcal{AG}$ has an individual task, which is defined as follows:

Definition 1 (*Individual task of an agent*) The task of an agent $i \in \mathcal{AG}$ is a tuple $\mathcal{T}^i = \langle \mathcal{I}^i, \Gamma^i \rangle$, where $\mathcal{I}^i$ describes the initial state of the task, and $\Gamma^i = \{\pi^i_1, \ldots, \pi^i_l\}$ is a finite set of plans that attain $\mathcal{T}^i$.

Our model is based on propositional STRIPS planning tasks. In this context, a plan $\pi^i \in \Gamma^i$ is defined as a sequence of *actions* $\pi^i = [a^i_0, \ldots, a^i_{m-1}]$. An action $a \in \pi^i$ is a triple $a = \langle pre(a), add(a), del(a) \rangle$: $pre(a)$ is the set of *preconditions* of a; $add(a)$ and $del(a)$ are two lists that denote the *positive* and *negative effects* of a, respectively. An action a is executable in a state S if $pre(a) \subseteq S$. Executing a in a state S yields a new state S', such that $S' = S \backslash del(a) \cup add(a)$.

The execution of a particular plan $\pi^i \in \Gamma^i$ reports agent i a reward or utility. In this planning scenario, every agent $i \in \mathcal{AG}$ wishes to execute the plan of Γ^i that reports the maximal utility.

Definition 2 (*Plan profile*) A plan profile is a collection of one plan per agent denoted with the tuple $\Pi = (\pi^1, \pi^2, \ldots, \pi^n)$, where $\pi^i \in \Gamma^i$ represents the individual plan choice of agent i.

The *actual utility* that a plan π^i reports to agent i depends on the concurrent execution of π^i with the rest of plans of the plan profile Π. Therefore, in this problem, the objective of an agent $i \in \mathcal{AG}$ is to select a plan π^i of Γ^i such that, when scheduled along with the rest of agents' choices in Π, it reports maximum utility to i.

Definition 3 (*Schedule of a plan*) The schedule of a plan $\pi^i \in \Gamma^i$ is a temporal sequence of actions that results from interleaving the actions in π^i with an arbitrary number of empty actions $\perp$. A plan schedule indicates the action of π^i to be executed at each time point.

We will denote by $\Upsilon^i = \{\psi_0^i, \psi_1^i, \ldots, \psi_x^i, \ldots\}$ the infinite set of all possible schedules of plan π^i. Given a particular schedule ψ_x^i, the *finish time* of the execution of ψ_x^i will be the time instant of the last action in ψ_x^i. In general, given two plan schedules $\psi_x^i, \psi_{x+1}^i \in \Upsilon^i$, the finish time of ψ_x^i is assumed to be prior or equal to the finish time of ψ_{x+1}^i. In the following, we will simply use the notation ψ^i to refer to any schedule of Υ^i.

The ideal schedule of a plan $\pi^i = [a_0^i, \ldots, a_{m-1}^i]$, ψ_0^i, consists in executing a_0^i in the state at $t = 0$ or initial state $\mathcal{I}^i$, and executing the subsequent actions of π^i at consecutive time instants. Thus, presumably, agent i will finish the execution of π^i at $t = m - 1$, the time of the last action scheduled in ψ_0^i (a_{m-1}^i). However, since agents execute their plans simultaneously in a common environment, *conflicts* that prevent agents from executing the ideal schedules of their preferred plans may arise. In case that a conflict compromises the ideal schedule ψ_0^i of a plan π^i, agent i may select an alternative schedule, ψ_x^i, which will comprise a number of empty actions $\perp$ that will help solve the conflict. The introduction of empty actions obviously entails a delay in the finish time of the plan execution, which in turn entails a loss of utility. The purpose of delaying actions is to avoid conflicts and ensure the executability or feasibility of a plan schedule.

Example 1 Given a plan $\pi^i = [a_0^i, a_1^i, a_2^i, a_3^i]$ of agent i, possible schedules for π^i are: $\psi_0^i = (a_0^i, a_1^i, a_2^i, a_3^i)$, $\psi_1^i = (\perp, a_0^i, a_1^i, a_2^i, a_3^i)$, $\psi_{10}^i = (a_0^i, a_1^i, \perp, a_2^i, \perp, a_3^i)$, $\psi_{19}^i = (a_0^i, a_1^i, \perp, a_2^i, \perp, \perp, a_3^i)$, etc. Particularly, ψ_0^i is the earliest plan execution of π^i (finishing at $t = 3$); ψ_1^i completes the execution of π^i at $t = 4$, ψ_{10}^i at $t = 5$ and ψ_{19}^i at $t = 6$.

The ultimate objective of the agents in $\mathcal{AG}$ is to come up with a combination of plan schedules (one per agent's plan) that is jointly executable. Since the plan choices of the agents may affect each other's utilities, the model proposed in this paper is a non-cooperative game-theoretic approach that solves the problem of finding a conflict-free (feasible) *schedule profile* which guarantees that the agents' plans of a plan profile Π are executable.

Definition 4 (*Schedule profile*) Given a plan profile $\Pi = (\pi^1, \pi^2, \ldots, \pi^n)$, a schedule profile of Π, s_Π, is a combination of one schedule per plan in Π; that is, $s_\Pi = (\psi^1, \psi^2, \ldots, \psi^n), \psi^i \in Y^i$.

A schedule profile $s_\Pi = (\psi^1, \psi^2, \ldots, \psi^n)$ induces a sequence of *joint actions*. A joint action is a tuple $A_t = \langle a^1, a^2, \ldots, a^n \rangle$, where a^i is the action of ψ^i scheduled at time instant t. In other words, A_t collects the actions of the plan schedules in s_Π (one action per agent in $\mathcal{AG}$) that agents intend to execute at time t.

Example 2 Given a schedule profile $s_\Pi = (\psi^1, \psi^2, \psi^3)$, $A_t = \langle a_2^1, \bot, a_3^3 \rangle$ is the joint action to be executed at time t, where agent 1 wants to execute its action a_2^1, agent 2 executes the empty action and agent 3 executes its action a_3^3.

Joint actions are applied over *joint states*. The initial joint state of the problem, $\mathcal{I}$, is defined as the union of the initial states of the agents in $\mathcal{AG}$; that is, $\mathcal{I} = \mathcal{I}^1 \cup \cdots \cup \mathcal{I}^n$. A joint action A_t is *executable* in a joint state S if no *conflict* arises at the time of executing the actions of A_t. We identify two types of conflicts in A_t:

- *Precondition conflict* One condition for A_t to be executable in a joint state S is that $\forall a \in \mathcal{A}_t, pre(a) \subseteq S$. It may happen that the execution of a joint action prior to A_t leads to a joint state S where some precondition of an action a of A_t does not hold. In this case, we say a precondition conflict occurs and, consequently, A_t is non-executable.
- *Mutually exclusive (mutex) conflict* This happens when two actions a and a' of A_t cannot be simultaneously executed at time t due to a mutex relationship as identified in the *GraphPlan* approach (Blum and Furst 1997). Particularly, two actions a and a' are said to be mutex if:
  - They have *inconsistent effects*; i.e., $add(a) \cap del(a') \neq \emptyset$.
  - They *interfere* with each other; i.e., $pre(a) \cap del(a') \neq \emptyset$.

Hence, if none of the above conflicts appears in A_t, then we say A_t is executable. The result of applying an executable joint action $A_t = \langle a^1, a^2, \ldots, a^n \rangle$ in a joint state S is a new joint state $S' = S \backslash (\bigcup_{i=1}^n del(a^i)) \cup (\bigcup_{i=1}^n add(a^i))$. When A_t is not executable, this may be fixed by delaying the action(s) in conflict through the introduction of empty actions in the corresponding schedule profile.

Definition 5 (*Feasible (conflict-free) schedule profile*) A schedule profile $s_\Pi = (\psi^1, \psi^2, \ldots, \psi^n)$ is *feasible* if and only if every joint action A_t of s_Π is executable.

Example 3 Let us assume that two agents 1 and 2 want to execute the plan profile $\Pi = (\pi^1 = [a_0^1, a_1^1, a_2^1], \pi^2 = [a_0^2, a_1^2, a_2^2])$; a possible schedule profile is $s_\Pi = (\psi^1 = (a_0^1, \bot, \bot, a_1^1, a_2^1), \psi^2 = (\bot, a_0^2, a_1^2, a_2^2))$. Additionally, s_Π is a feasible schedule profile if every joint action is executable (the joint actions for s_Π are $A_0 = \langle a_0^1, \bot \rangle$, $A_1 = \langle \bot, a_0^2 \rangle$, $A_2 = \langle \bot, a_1^2 \rangle$, $A_3 = \langle a_1^1, a_2^2 \rangle$, $A_4 = \langle a_2^1 \rangle$).

Given a plan profile $\Pi = (\pi^1, \ldots, \pi^n)$ and an associated schedule profile $s_\Pi = (\psi^1, \ldots, \psi^i, \ldots, \psi^n)$, the *maximum number of empty actions* in the schedule ψ^i of an agent i, is limited by the sum of the actions of the other agents' plans in Π, denoted by λ^i_Π:

$$\lambda^i_\Pi = \sum_{j \neq i}^{\pi^j \in \Pi} |\pi^j|. \tag{1}$$

If we consider a problem where the number of schedules of a plan π^i associated to a plan profile Π is not limited by λ^i_Π, it is possible to find additional schedule profiles by adding more empty actions. Any additional schedule profile of a non-limited problem will report less utility to (at least) some agent i because it would include a number of empty actions larger than λ^i_Π. Therefore, we can conclude that the additional schedule profiles that can be formed in a non-limited planning problem are weakly Pareto dominated by the schedule profiles of the original problem limited by λ^i_Π.

Example 4 Given a plan profile $\Pi = (\pi^1 = [a^1_0, a^1_1, a^1_2], \pi^2 = [a^2_0, a^2_1, a^2_2])$, $\lambda^i_\Pi = 3$ for both agents, $i = \{1, 2\}$. A schedule with more than 3 empty actions for any agent is useless since the maximum number of empty actions necessary to address the conflicts is 3. For instance, the schedule profile $s_\Pi = (\psi^1 = (a^1_0, a^1_1, a^1_2), \psi^2 = (\bot, \bot, \bot, a^2_0, a^2_1, a^2_2))$ introduces 3 empty actions in ψ^2 and so all the joint actions in s_Π include a single action ($A_0 = \langle a^1_0, \bot\rangle$, $A_1 = \langle a^1_1, \bot\rangle$, $A_2 = \langle a^1_2, \bot\rangle$, $A_3 = \langle \bot, a^2_0\rangle$, $A_4 = \langle \bot, a^2_1\rangle$ and $A_5 = \langle \bot, a^2_2\rangle$).

Thus, given a plan profile Π, if a feasible schedule profile cannot be obtained by means of λ^i_Π empty actions for every agent $i \in AG$, introducing more empty actions than λ^i_Π in the plan schedule of any agent will not yield a feasible schedule profile for Π. In this case, we say that all the schedule profiles for Π are *infeasible*. Particularly, an infeasible schedule profile is due to a precondition conflict because mutex conflicts are always solvable by introducing empty actions. However, even introducing λ^i_Π empty actions in all the schedule profiles, it may not be possible to find a joint state S in which all the preconditions of an action in a joint action A_t are satisfied.

Definition 6 (*Utility of a plan schedule*) The utility function $u^i : Y^i \to \mathbb{R}$ returns the utility of a schedule ψ^i of a plan π^i for agent i. For a given π^i, the difference of utility of two plan schedules ψ^i_x and $\psi^i_{x'}$, $x' > x$, is given only by the difference in their finish execution times. The later the finish time, the less utility. Consequently, by default, the ideal schedule ψ^i_0 of a plan π^i is the schedule that reports agent i the maximal utility and the rest of schedules of Y^i will have a lower utility accordingly to their finish time. An infeasible (non-executable) schedule profile reports each agent $i \in \mathcal{AG}$ a utility $u^i = -\infty$.

In this section, we have introduced and formalized all the components that are necessary for the specification of our game-theoretic approach FENOCOP.

3 Overview of FENOCOP

FENOCOP (Fair Equilibria in NOn-COoperative Planning) is our computational framework for the resolution of conflicts in non-cooperative MAP. As described in Sect. 2, the problem we aim to solve involves a set of self-interested planning agents, $\mathcal{AG}$, where each agent i independently works on its individual task $\mathcal{T}^i$ by calculating a finite collection Γ^i of plans of different utility that solve $\mathcal{T}^i$.

In a game-theoretic context like FENOCOP, the plans of Γ^i represent the different *strategies* of agent i to accomplish its task; i.e., the options or alternatives that the agent can choose in the game. As commented in Definition 2, the actual utility that a plan or strategy π^i reports to agent i will be subject to the schedule of π^i with the rest of agents' strategies in a plan profile Π. In the following, we will refer to a plan or a strategy interchangeably.

Every agent i wishes to execute the ideal plan schedule ψ_0^i of the maximum utility plan π^i. On the other hand, given that this is a non-strictly competitive environment, agent i also wants to make its course of action ψ_0^i compatible with the rest of the agents' proposals of a plan profile and thus ensure that every agent is able to execute a plan that achieves its task.

Conflicts may appear when the plan schedules of multiple agents are put together to execution in a shared environment. A conflict between two particular plan schedules ψ_x^i and ψ_y^j entails that either agent i or agent j cannot execute its plan. When this happens, one or both agents must switch to a different schedule so as to avoid the interference. Assuming agent i selects a new schedule $\psi_{x'}^i$, some actions of ψ_x^i will be delayed in $\psi_{x'}^i$ through the inclusion of empty actions in order to solve the conflict, which in turn implies a delay in the finish time of the execution of agent i. If the new schedule $\psi_{x'}^i$ entails a significant loss of utility, agent i may select a different plan from Γ^i that, when scheduled with the rest of agents' plans, brings higher utility. Hence, agents must find together a feasible schedule profile s_Π that ensures the executability of the plans while satisfying the private interests (utility) of the participants.

A rational way of solving the conflicts that arise among a set of self-interested agents with potentially conflicting interests implies modelling the problem as a non-cooperative game. FENOCOP is a non-cooperative two-game mechanism guided by a top-level game called *General Game* (GG), which leverages an internal game called *Scheduling Game* (SG). Particularly, the GG of FENOCOP works as follows:

1. It generates the $\Gamma^1 \times \cdots \times \Gamma^n$ plan profiles that result from combining the strategies of the n agents in $\mathcal{AG}$.
2. For every plan profile Π, the GG calls the SG to calculate a schedule profile s_Π. The outcome s_Π returned by the SG holds the following properties:

   (a) it is a *stable outcome* from which no agent is willing to deviate; that is, it is a Nash equilibrium (NE) solution

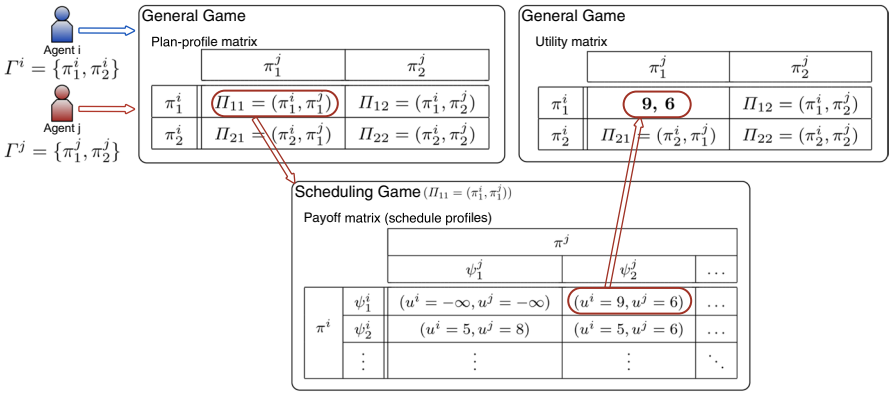


Fig. 1 An iteration of FENOCOP

(b) it is a Pareto Optimal (PO) outcome and as such it outperforms any Pareto-inefficient NE solution
(c) it is a fair solution that guarantees a balance among the agents' utilities

3. From the set of feasible or infeasible schedule profiles $\{s_{\Pi_1}, s_{\Pi_2}, \ldots\}$ calculated by the SG, the GG returns a stable s^*_{Π}, a NE solution that guarantees (1) the plan schedules of all the agents in $\mathcal{AG}$ are executable; and (2) no agent will deviate from its course of action in s^*_{Π} because no agent can do better by unilaterally changing its strategy. In the case that the schedule profile for every plan profile is infeasible then the task is *unsolvable*. That is, there is not an executable combination of the agents' strategies.

Since agents operate in a non-strictly competitive environment, the GG is designed as a *general-sum game* or non-zero sum game (Shoham and Leyton-Brown 2009; Osborne and Rubinstein 1994). In this type of games there can be win–win situations because, unlike competitive games, general-sum games feature situations where one decision agent's gain (or loss) does not necessarily result in the other decision agents' loss (or gain).

Figure 1 shows graphically an example of FENOCOP for two self-interested agents i and j, each having two strategies $\Gamma^i = \{\pi_1^i, \pi_2^i\}$ and $\Gamma^j = \{\pi_1^j, \pi_2^j\}$, respectively. The two upper matrices represent the GG in normal or strategic form. This form is given by the two sets Γ^i and Γ^j of agents' strategies (*plan-profile* matrix on the left), and two real-valued utility functions defined on $\Gamma^i \times \Gamma^j$, representing the payoffs to both agents (*utility* matrix on the right). The bottom matrix represents the internal *Scheduling Game*. The SG is actually the game that computes a stable (NE), PO and fair schedule profile s_{Π} for each plan profile Π. Thus, for each cell in the plan-profile matrix, the GG invokes the SG, which returns the utility received by each agent with s_{Π}. For instance, in Fig. 1, the SG is called to compute a feasible schedule profile for Π_{11}, selecting the outcome $s_{\Pi_{11}} = \{\psi_1^i, \psi_2^j\}$, which is then stored in the utility matrix of the GG. Note that the top left cell of the payoff matrix

denotes an infeasible schedule that reports a utility value $-\infty$ to both agents. Once the utility values of all plan profiles are stored in the utility matrix, the GG returns a stable solution s^*_{Π}.

The key novelty of FENOCOP with respect to other game-theoretic approaches like Bowling et al. (2003) and Larbi et al. (2007) is the introduction of a planning algorithm in the form of a game, the *Scheduling Game*, to compute the payoffs of the plan profiles. Specifically, these two works propose a framework equivalent to our top-level GG, but there is no indication on how to actually achieve a feasible schedule profile that accommodates the plans of all the agents.

4 The General Game

The top-level game of FENOCOP, called the General Game (GG), aims to select a stable (NE) schedule profile among the combinations of the agents' strategies. The GG is then modelled as a non-cooperative general-sum game represented in the *normal-form*. This type of game is defined by its *players* (agents), the *strategies* or plans among which they can choose, and the *payoffs* they will each receive for a given strategy. Formally, the GG is defined as follows:

Definition 7 *General Game* (GG) The GG is a general-sum game with an associated triple $(\mathcal{AG}, \Gamma, u)$, where:

- $\mathcal{AG} = \{1, \ldots, n\}$ is the set of n rational and self-interested agents, the *players* of the GG.
- $\Gamma = \Gamma^1 \times \cdots \times \Gamma^n$ represents a finite set of plan profiles or combinations of the agents' strategies. A plan profile is a set of plans of the form $\Pi = (\pi^1, \pi^2, \ldots, \pi^n)$, where $\pi^i \in \Gamma^i$ for each agent $i \in \mathcal{AG}$.
- $u = (u^1, \ldots, u^n)$ is a set of utility functions, where $u^i : \boldsymbol{\Upsilon}^{\mathbf{i}} \rightarrow \mathbb{R}$ is the real-valued *payoff* function for agent i (as specified in Definition 6). Particularly, let $\Upsilon^i(\pi^i)$ be all possible schedules ψ^i (see Definition 3) for plan π^i. Then, $\boldsymbol{\Upsilon}^{\mathbf{i}} = \bigcup_{\forall \pi^i \in \Gamma^i} \Upsilon^i(\pi^i)$ is the set of all plan schedules for agent i in the GG. So $u^i(\psi^i)$ is the utility that a particular schedule $\psi^i \in \Upsilon^i(\pi^i)$ of plan $\pi^i \in \Gamma^i$ reports to agent i.

We must note that the payoff that a particular strategy or plan π^i reports to agent i depends on how π^i is combined with the rest of plans of the plan profile Π; i.e., the actual utility is given by the schedule profile $s_{\Pi} = (\psi^1, \ldots, \psi^i, \ldots, \psi^n)$ returned by the SG. s_{Π} will determine the specific plan schedule ψ^i for each agent i, which in turn determines the utility obtained by agent i in the plan combination, $u^i(\psi^i)$.

In order to create the utility matrix of the GG, agents launch $\Gamma^1 \times \cdots \times \Gamma^n$ instances of the SG, one per plan profile Π, and the SG computes a schedule profile s_{Π} along with the utility that s_{Π} reports to each agent. Once all the agents' utilities are in place, solving the GG means to compute the final solution s^*_{Π}. This schedule profile constitutes a NE stable solution from which no agent will benefit from invalidating another agent's plan schedule.

5 The Scheduling Game

As described in Sect. 3, the Scheduling Game (SG) is invoked for each combination of strategies or plan profile $\Pi = (\pi^1, \ldots, \pi^n)$ of the GG in order to retrieve a feasible (executable) schedule profile s_Π that satisfies stability, Pareto optimality and fairness, if such a schedule profile exists. The SG is structured around the following two stages:

1. *Synthesis of schedule profiles* The SG computes the schedule profiles that coordinate the agents' strategies of the plan profile Π. The resulting payoff matrix (see bottom matrix in Fig. 1) contains the utilities that the schedule profiles report to each participant.
2. *Schedule profile selection* Agents solve the game in order to select a stable, PO and fair outcome.

In the first stage of the SG, agents coordinate their plans to guarantee that they are executable in a shared environment. Given a schedule profile s_Π, agents verify that each joint action $A_t \in s_\Pi$ is executable; otherwise, empty actions ($\perp$) are introduced in A_t in order to solve the conflicts that prevent A_t from being executable in a state S. The introduction of an empty action defers the execution of an action of A_t to a later time step $t' > t$. The number of empty actions that an agent i can introduce in a plan schedule $\psi^i \in s_\Pi$ is delimited by λ^i_Π, and hence, there is a finite number of schedule profiles for any given plan profile Π.

After synthesizing the schedule profiles for Π, the self-interested agents jointly select an outcome that maximizes their utilities by taking into account the plan schedules of the other participants. Since a conflict between a subset of plan schedules renders the whole schedule profile infeasible, every agent i receives a utility $u^i(\psi^i) = -\infty$ for its plan schedule ψ^i in an infeasible schedule profile. For this reason, we can affirm that the loss of utility of an agent is not the utility gain of the other agents; and so, the SG is a non-strictly competitive problem modelled as a general-sum game. Formally:

Definition 8 *Scheduling Game* (SG) The SG is a general-sum game defined by an associated tuple $(\Pi, \mathcal{AG}, \Psi_\Pi, u)$, where:

- $\Pi = (\pi^1, \ldots, \pi^n)$ is a combination of plans or *plan profile* for which the SG must find an executable schedule profile s_Π.
- $\mathcal{AG} = \{1, \ldots, n\}$ is the set of n rational and self-interested agents or *players*.
- $\Psi_\Pi = \Psi^1_\Pi \times \cdots \times \Psi^n_\Pi$ is the set of schedule profiles for the plan profile $\Pi = (\pi^1, \ldots, \pi^n)$ represented in the payoff matrix (see Fig. 1), where each agent i has a finite set of *strategies* $\Psi^i_\Pi = \{\psi^i_0, \psi^i_1, \ldots, \psi^i_k\}$, where $\Psi^i_\Pi \subset \Upsilon^i$, the possible schedules of its plan $\pi^i \in \Pi$.
- $u = (u^1, \ldots, u^n)$ where $u^i \colon \Upsilon^i \to \mathbb{R}$ is a real-valued *payoff function* for agent i. $u^i(\psi^i)$ is defined as the utility of the schedule $\psi^i \in \Psi^i_\Pi$ when executed in a

schedule profile $s_{\Pi} = (\psi^1, \ldots, \psi^{i-1}, \psi^i, \psi^{i+1}, \ldots, \psi^n)$. If s_{Π} is infeasible, then $u^i(\psi^i) = -\infty$ for all agents.

The set of plan schedules, Ψ^i_{Π}, that agent i uses to combine its plan $\pi^i \in \Pi$ with the rest of plans of Π is a finite subset of Y^i. Considering, as stated in Eq. 1, that the number of empty actions of any plan schedule ψ^i is limited by λ^i_{Π}, the number of plan schedules in Ψ^i_{Π} is given by all the combinations that can be formed with the actions in π^i and up to λ^i_{Π} empty actions.

6 Solving the Scheduling Game

This section is devoted to explain two different solving algorithms for the SG. First, we motivate the relevance of three well-known *solution concepts* in non-cooperative game-theory; namely, Nash equilibrium, Pareto Optimality and fairness. Next, in Sect. 6.2, we present two key properties of the SG that will strongly contribute to guarantee the solution concepts of a schedule profile. The following two subsections explain the *normal-form* and *extensive-form* SG algorithms, respectively. Both algorithms follow the two stages of the SG presented in Sect. 5 and compute solutions that meet the three aforementioned concepts.

6.1 Solution Concepts in Non-cooperative Games

We aim for finding equilibrium solutions that represent a stable joint plan schedule for all the agents. Since multiple equilibrium solutions can be found, we further apply an optimality criteria to filter out those solutions which do not comparatively bring any utility improvement to any of the agents. Subsequently, in case various outcomes still remain, we apply a further criteria of fairness so as to promote the individual satisfaction of the agents with a given solution.

Nash equilibrium A Nash equilibrium (NE) or *stable* solution reflects the best response of an agent taking into account the responses of the rest of agents. In an equilibrium, no agent can benefit from deviating unilaterally from a joint solution. In the SG, a NE outcome is a schedule profile in which an agent cannot improve its utility unless another agent changes its plan schedule. Since an SG can have several NE outcomes (feasible or infeasible schedule profiles), we introduce a second criterion to choose among them, Pareto optimality.

Pareto Optimality We promote schedule profile solutions for which we know that there is no other schedule profile that is at least as good for all agents, and strictly better for one. This best equilibrium schedule profile is called a Pareto Optimal (PO) schedule profile and reflects a situation where no agent can be better off without making at least one agent worse off.

Fairness Fairness is a criterion that applies to the satisfaction of the agents with their individual utilities. Among the many existing fairness criteria, the *egalitarian* principle in ethical theory asserts that all the individuals should enjoy equal benefits from the society (Rawls 1971). As long as there is a positive

Table 1 SG example in normal-form for two agents

	ψ_0^j	ψ_1^j
ψ_0^i	$-\infty, -\infty$	*7, 9*
ψ_1^i	***8, 6***	**7, 6**

NE outcomes in bold, PO outcomes in italics, and fair outcome underlined

trade-off between the utility of different individuals, the egalitarian principle leads to the same social choices as the *maxmin* principle, which maximizes the utility of the most unfortunate individuals of a society (*egalitarian social welfare*) (Myerson 1981). According to the *maxmin* principle, an outcome is fair if it maximizes the minimum utility received by any agent; i.e., the least satisfied agent is as satisfied as possible. This way, a resource allocation amongst agents in multi-agent systems is considered fair if it is egalitarian (Chevaleyre et al. 2006; Endriss et al. 2006).

In this work, the application of fairness lies in analyzing the schedule profiles in terms of the individual satisfaction of the participants in order to ensure a proper balance of the agents' utilities. In the context of the SG, egalitarian social welfare guarantees that the least satisfied agent has the minimum possible delay. Given a set of NE and PO schedule profiles for a plan profile Π, denoted by $\Omega_\Pi \subseteq \Psi_\Pi$, we define a *fair schedule profile* $\widehat{s_\Pi} \in \Omega_\Pi$ as the schedule profile that results from the application of the max-min utility criterion over Ω_Π:

$$\widehat{s_\Pi} = \arg\max_{s_\Pi \in \Omega_\Pi} \left(\min_{i \in \mathcal{AG}} u^i(s_\Pi) \right). \tag{2}$$

The schedule profile that maximizes the utility of the agent which has less utility among the schedule profiles of Ω_Π is selected as the fair solution $\widehat{s_\Pi}$ of the SG. More than one fair solution can be found if several schedule profiles with the same max–min utility exist in Ω_Π.

Let us introduce an example to illustrate how the presented solution concepts are applied to the SG.

Example Assume we have a 2-agent (agent i and agent j) SG with 4 possible outcomes (schedule profiles) as shown in Table 1. A cell like ψ_0^i, ψ_0^j is a schedule profile that represents the strategy 0 of agent i when combined with the strategy 0 of agent j and the values in the cell are the utilities that the agents receive with this schedule profile. The outcome of the schedules/strategies are:

- the outcome of the schedule profile ψ_0^i, ψ_0^j reports utilities $(-\infty, -\infty)$ to agent i and j, respectively
- the outcome of the schedule profile ψ_0^i, ψ_1^j reports utilities (7, 9) to agent i and j, respectively
- the outcome of the schedule profile ψ_1^i, ψ_0^j reports utilities (8, 6) to agent i and j, respectively

- the outcome of the schedule profile ψ_1^i, ψ_1^j reports utilities (7, 6) to agent i and j, respectively

There are three NE outcomes in this SG (in bold in Table 1); that is, all the outcomes are NE except the one with utilities $(-\infty, -\infty)$. We can filter the three NE solutions by applying additional criteria. Hence, if we apply the PO criterion we will end up with two solutions (in italics in Table 1), those with utilities (7, 9) and (8, 6), since the outcome (7, 6) is Pareto dominated by both of them. Finally, it is still possible to further filter the solutions that are both NE and PO by applying the concept of fairness. In this case, the outcome (7, 9) is a fair solution (underlined in Table 1) while (8, 6) is not. Note that the most harmed agent in the outcome (7, 9) is agent i with a utility of 7 while the most harmed in outcome (8, 6) is agent j but with a utility of 6. Thus, applying fairness over these two solutions returns the outcome (7, 9) because the most harmed agent in this solution (i) is not as harmed as the most harmed agent (j) in the other outcome. Consequently, the schedule profile with utilities (7, 9) is the solution of this SG because it is the only one that meets the three criteria, namely: NE, PO, and fair. In case of more than one outcome that meets the three criteria, a random solution would be chosen.

6.2 Properties of the Scheduling Game

As a first observation, we must note that the utility of the SG is only influenced by the conflicts and the empty actions. Additionally, an agent only impacts the utility of another agent through the conflict handling. The SG features two properties that can be enunciated as follows:

- *Monotonicity* An SG is said to be *monotonic* if the utility $u^i(\psi^i)$ of any feasible plan schedule $\psi^i \in \Psi_{\Pi}^i$ decreases according to the number of empty actions $\perp$ in ψ^i. In other words, given two plan schedules ψ_x^i and ψ_{x+1}^i, then $u^i(\psi_x^i) \geq u^i(\psi_{x+1}^i)$. In Definition 6, we stated that the loss of utility of a plan schedule is only dependent on the finish time of the schedule. Consequently, every SG is monotonic.
- *Order* An SG is *ordered* if the strategies of the agents are ordered by decreasing utility in the game. More precisely, if the game is monotonic, for an agent $i \in \mathcal{AG}$, the strategies of Ψ_{Π}^i are ordered from 0 to λ_{Π}^i empty actions. This property is useful to reduce computation time of the algorithms by pruning.

Proposition 1 *In an* SG, *if the schedule profile formed by the ideal schedule* ψ_0^i *of each agent* i *is feasible, then this schedule profile is the only outcome of the* SG *which is both NE and PO.*

Proof In the absence of conflicts, all agents have the highest utility $u^i(\psi_0^i)$ with their ideal schedule since there are no empty actions. This schedule profile will also

Table 2 SG example 2 in normal-form for two agents

	ψ_0^2	ψ_1^2	ψ_2^2
ψ_0^1	$-\infty, -\infty$	$-\infty, -\infty$	$-\infty, -\infty$
ψ_1^1	$-\infty, -\infty$	$-\infty, -\infty$	**9, 8**
ψ_2^1	$-\infty, -\infty$	**8, 9**	8, 8

NE outcomes in bold

be unique because any other schedule profile will have less utility for at least one agent. □

A feasible schedule profile s_{Π} with maximum utility for an agent i is PO if, for any other feasible schedule profile s'_{Π} with the maximum utility for the agent i, the utility of the other agents is not higher than their utility in s_{Π}. In this situation, all the agents in $\mathcal{AG}$ are in best response; and thus, s_{Π} is a PO NE schedule profile.

Theorem 1 *In an* SG, *any PO schedule profile is a NE.*

Proof By contradiction, suppose a change in strategy of an agent j from a PO profile increases its utility: it must be reducing its empty actions because a conflict cannot be introduced since then its utility is decreased to $-\infty$. So the utility of no other agent is affected, while j's utility is improved; this is a contradiction with the profile being PO. Hence j cannot change its strategy to increase its utility, so the PO schedule profile is also a NE. □

In a monotonic SG we thus only need to seek PO outcomes because a PO outcome s_{Π} is always a NE, which guarantees that no agent will be willing to deviate from its strategy in s_{Π}. Therefore, any potential solution of the SG is a PO NE schedule profile.

In the SG, not every NE schedule profile is necessarily PO and it can actually be an infeasible outcome. In the example of Table 2, the top left cell is a NE with utility $u^i(\psi^i) = -\infty$ for both agents. This happens because there is no better response for those strategies (all the cells that involve the optimal strategy ψ_0^i for any agent i are infeasible outcomes with $u^i(\psi_0^i) = -\infty$). For this reason, a solution of the SG must not only be a NE, but also PO.

Corollary 1 *If there is at least one feasible schedule profile for a monotonic* SG, *there will be at least a PO NE solution for the game.*

The definition of Pareto optimality establishes that a schedule profile $s_{\Pi} = (\psi^1, \ldots, \psi^n)$ is PO if it is not Pareto dominated by any other schedule profile $s'_{\Pi} = (\psi^{1'}, \ldots, \psi^{n'})$; that is, $u^i(\psi^i) \geq u^i(\psi^{i'})$, $\forall i \in \mathcal{AG}$ and $u^i(\psi^i) > u^i(\psi^{i'})$ for some $i \in \mathcal{AG}$. From this definition, it can be drawn that every game must have at least one such optimum (Shoham and Leyton-Brown 2009, Chapter 3). Given that

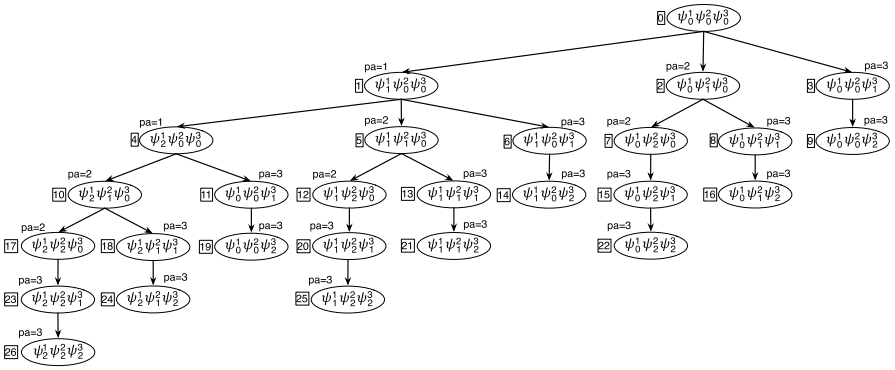


Fig. 2 BFS tree with all the schedule profiles for 3 agents and 3 schedules per agent

any PO outcome is a NE according to Theorem 1, if at least one feasible schedule profile exists, there is a PO NE solution for the SG.

6.3 Normal-Form SG Algorithm

Given an ordered monotonic SG, the normal-form algorithm obtains all fair PO NE feasible schedule profiles (solutions) of the game. The algorithm applies a *breadth-first search* (BFS) where each node of the search tree represents a specific schedule profile $s_{\Pi} = (\psi^1, \ldots, \psi^n)$. The algorithm can be summarized as follows:

1. The root node of the tree is a schedule profile that contains the ideal or highest-utility plan schedule for each agent; i.e., $s_{\Pi} = (\psi_0^1, \ldots, \psi_0^n)$.
2. The feasibility of a schedule profile is checked at the time of expanding the node. If s_{Π} results infeasible, its children nodes are generated. A successor node changes the plan schedule of a single agent in s_{Π} by its next plan schedule in decreasing order of utility; for instance, the children of $(\psi_0^1, \ldots, \psi_0^n)$ are $(\psi_1^1, \psi_0^2, \ldots, \psi_0^n)$, $(\psi_0^1, \psi_1^2, \ldots, \psi_0^n)$ ...$(\psi_0^1, \psi_0^2, \ldots, \psi_1^n)$. In case that s_{Π} is feasible, the algorithm applies the PO and fairness conditions over s_{Π} in order to check whether or not s_{Π} Pareto dominates and is fairer than any previous feasible node.
3. The search concludes when there are no more nodes to be expanded. At this point, the algorithm returns the set $\widehat{s_{\Pi}}$, which comprises the nodes of the tree that represent NE, PO and fair solutions.

Figure 2 shows an illustrative example of the BFS tree. This example includes three agents (named 1, 2, and 3), each having three different plan schedules (ψ_0^i, ψ_1^i and ψ_2^i, for each agent i). The numbers in squares are the node identifiers and the *pa* labels indicate the *pivot agent* of the node (see details below).

Algorithm 1 details the normal-form SG procedure. The initial schedule profile, consisting of the ideal schedule of each agent, is added to a queue (lines 1–3). The parameter $s_{\Pi}.pivotAgent$ represents the agent whose plan schedule is changed in s_{Π} with respect to its parent node. $s_{\Pi}.pivotAgent$ is used to prevent the generation

of repeated or Pareto dominated nodes. The *maxMinBound* parameter stores the maxmin utility of $\widehat{s_\Pi}$ for fairness purposes, and $maxUAg^i$ stores the maximum utility of agent i. Both parameters are initialized to $-\infty$ (lines 4–6).

The *while* loop of the algorithm iterates until the queue of schedule profiles is empty. An iteration of the procedure extracts a schedule profile s_Π from the queue and verifies its *fairness*. s_Π is fair if the minimum utility obtained by an agent in s_Π ($min\ u^j(\psi^j)$, where $\psi^j \in s_\Pi$) is greater or equal than *maxMinBound* (line 9). Otherwise, s_Π is discarded.

Next, the *feasibility* of s_Π is checked by means of the `conflicts`(s_Π) function (line 10). Depending on the result of this verification, different tasks are performed:

- *s_Π is feasible (lines 11–17)* The *Pareto optimality* of s_Π is analyzed by checking that $u^i(s_\Pi) > maxUAg^i$ for at least one agent i in $\mathcal{AG}$ (line 12). If this condition holds, s_Π is confirmed as PO because the agents' schedules are processed in decreasing utility order. Otherwise, s_Π is discarded. If s_Π is fairer than the schedule profiles in $\widehat{s_\Pi}$, (that is, $min\ u^j(s_\Pi) > maxMinBound$), s_Π is stored as the single fair solution in $\widehat{s_\Pi}$. Otherwise, s_Π is added to the $\widehat{s_\Pi}$ set (lines 14–17).
- *s_Π is infeasible (lines 18–22)* The successor nodes of s_Π are generated and added to the queue. A successor node changes the plan schedule ψ^i_x of an agent i by ψ^i_{x+1}, the next schedule of the agent in decreasing order of utility. The *for* loop (lines 19–22) iterates (using the index i) from the pivot agent (stored in $s_\Pi.pivotAgent$) to agent n, generating a total of $n - i + 1$ successor nodes.

The successor nodes of a feasible schedule profile s_Π are not generated because they would be Pareto dominated by s_Π. This conclusion is easily drawn by the monotonicity property, which ensures that the utility of the pivot agent in a successor node is always lower or equal than the utility of its parent schedule profile while the plan schedules of the rest of agents are kept unchanged. All in all, Pareto dominance allows for a meaningful pruning of the BFS search tree.

Algorithm 1: Normal-form SG algorithm

```
1  s_Π = ⋃_{i=1}^{n} ψ_0^i;
2  s_Π.pivotAgent = 1;
3  add s_Π to queue;
4  maxMinBound = −∞;
5  for i=1,...,n do
6      maxUAg^i = −∞;
7  while ¬(empty queue) do
8      extract s_Π from queue;
9      if min u^j(ψ^j), ψ^j ∈ s_Π ≥ maxMinBound then
10         if ¬ conflicts(s_Π) then
11             for i=1,...,n do
12                 if u^i(s_Π) > maxUAg^i then
13                     maxUAg^i = u^i(s_Π);
14                     if min u^j(s_Π) > maxMinBound then
15                         maxMinBound = min u^j(s_Π);
16                         ŝ_Π = ∅;
17                     add s_Π to ŝ_Π; break;
18         else
19             for i=s_Π.pivotAgent,...,n do
20                 s'_Π = (ψ^1, ..., ψ^i_{x+1}, ..., ψ^n); ψ^i_x ∈ s_Π; ψ^i_{x+1} ∈ Ψ^i;
21                 s'_Π.pivotAgent = i;
22                 add s'_Π to queue;
23 return ŝ_Π;
```

6.3.1 Complexity of the Normal-Form SG Algorithm

The normal-form algorithm develops a search tree with a maximal branching factor of $|\mathcal{AG}|$. For instance, in the example of Fig. 2, which includes 3 agents, up to three successors per schedule profile are generated (excluding repeated nodes). The maximal depth of the search tree is determined by the number of schedule profiles for Π, $|\Psi_\Pi|$.

Given the previous considerations, the normal-form SG algorithm presents an exponential cost that can be denoted as $\mathcal{O}(|\mathcal{AG}|^{|\Psi_\Pi|})$. In practical terms, several mechanisms are applied in order to alleviate the complexity of the algorithm, as illustrated in Fig. 2:

- The successors of a feasible and PO schedule profile are never generated because the order and monotonicity properties of the SG ensure that all the successors of a feasible PO schedule profile are always Pareto dominated by their parents.
- Cycles in the search tree are controlled in order to prevent the appearance of repeated nodes. For instance, in Fig. 2, the node $(\psi_1^1, \psi_1^2, \psi_0^3)$ does not appear as a successor of node 2 because it is already included in the subtree of node 1 (see node 5 in Fig. 2).
- Pareto dominance is also checked among nodes of different subtrees. Let us suppose that the node 1 of Fig. 2, $(\psi_1^1, \psi_0^2, \psi_0^3)$, is a feasible schedule profile, in which case the subtree of this node would not be generated. The schedule

$(\psi_1^1, \psi_1^2, \psi_0^3)$, which is Pareto dominated by node 1, would not either be included in the subtree of node 2, $(\psi_0^1, \psi_1^2, \psi_0^3)$, because the generation of the successors of a node s_{Π} goes from $s_{\Pi}.pivotAgent$ to n. Since the pivot agent of node 2 is agent 2, its two successors represent a change in the plan schedules of agent 2 and 3, respectively, leaving the schedule of agent 1 unchanged; i.e., ψ_0^1. Consequently, no successor with ψ_1^1 will be generated as a descendent of node 2 even though the subtree of node 1 is not created.

Despite the usage of pruning mechanisms in the BFS tree, the normal-form SG algorithm is a costly procedure that entails exploring most of the schedule profiles in Ψ_{Π} in order to find a feasible PO and fair solution. Moreover, the branching factor of the search tree is determined by $|\mathcal{AG}|$, which significantly impacts the performance of the algorithm when the number of agents is increased.

6.4 Extensive-Form SG Algorithm

In this section, we propose a completely different approach to solve the SG which relies in modelling the problem as an extensive-form game (Shoham and Leyton-Brown 2009, Chapter 5). The extensive-form algorithm poses the SG as a multi-round sequential game where agents play in turns and incrementally build a feasible schedule profile. This algorithm, which also obtains all fair PO and NE solutions, draws upon a former algorithm presented in Jordán and Onaindía (2015). The work in Jordán and Onaindía (2015), a theoretical framework that features an extensive-form game, applies the *Subgame Perfect Equilibrium* (SPE) solution concept (Shoham and Leyton-Brown 2009, Chapter 5). An SPE solution is a refinement of a NE solution that finds the schedule profiles that are NE for any subgame of the game. Informally speaking, the SPE eliminates the branches of an extensive-form tree which would involve any player making a move that is not credible (because it is not optimal) from that node. However, the present proposal applies a more advanced concept of solution different from the SPE. The solution concept of the current proposal searches for efficient schedule profiles (Pareto optimality property, which implies NE by Theorem 1) that present an equitable distribution of the loss of utility caused by the existence of conflicts (fairness property).

The extensive-form game is based on a binary tree where agents incrementally generate the schedule profiles for Π action by action. Thus, the branching factor of the tree remains constant regardless of the number of participating agents. This algorithm executes a *depth-first search* (DFS) where a tree node represents the action choice of an agent given the actions introduced in its predecessor nodes.

Figure 3 presents an illustrative example of the tree which includes two different agents, $\mathcal{AG} = \{1, 2\}$. The top left square represents the plan profile of this particular SG, $\Pi = (\pi^1 = [a_1^1, a_2^1], \pi^2 = [a_1^2, a_2^2])$, where preconditions and effects of the actions are shown above and below the nodes, respectively. The nodes of the tree are numbered according to the order in which they are visited by the DFS search. The nodes introduced by agent 1 are depicted in a darker color than those of agent

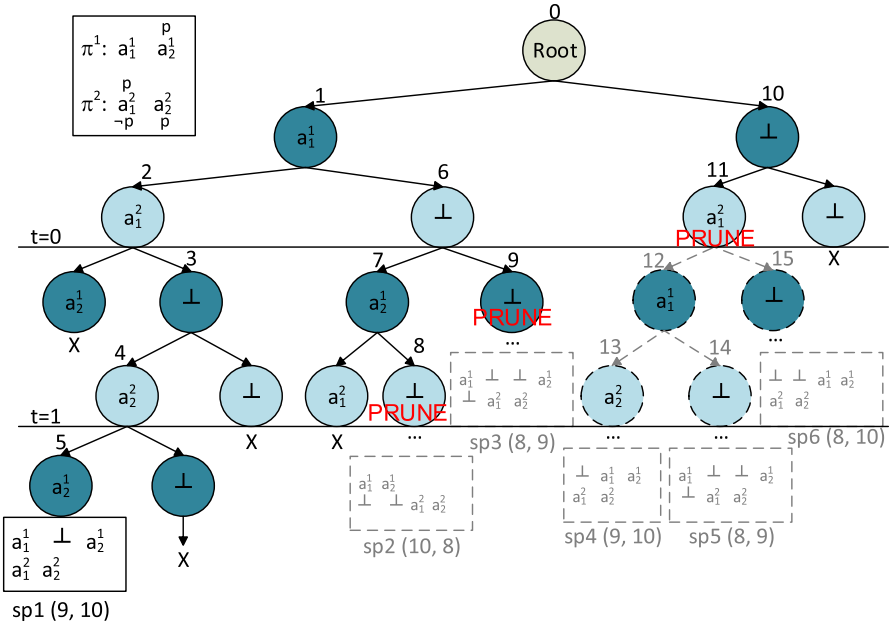


Fig. 3 SG extensive-form tree example

2. Using this example, we can summarize the behavior of the extensive-form SG algorithm as follows:

1. From the root node, agent 1 generates two successors that represent its possible initial choices, either introducing the first action of its plan, $a_1^1 \in \pi^1$, or an empty action $\perp$ (nodes 1 and 10). At the next level, agent 2 expands node 1 and generates two successors with actions $a_1^2 \in \pi^2$ and $\perp$ (nodes 2 and 6). Next, agent 1 responds by expanding node 2, incorporating actions a_2^1 and $\perp$, respectively. Specifically, the lines labelled as $t = 0$, $t = 1$, etc., delimit the levels of the game; that is, the first game level comprises the nodes up to $t = 0$, which represent the choices of the joint action A_0; the third and fourth level of the tree represent the second game level ($t = 1$) whose nodes represent the formation of A_1; and so on.
2. For each node, the presence or absence of conflicts is verified to ensure that only feasible schedule profiles are generated. In Fig. 3, a precondition conflict is detected when agent 1 expands node 2 to insert the action a_2^1 (the precondition $p \in pre(a_2^1)$ does not hold in the corresponding joint state because of the negative effect $\neg p$ of a_1^2). This node is discarded because it does not yield a feasible schedule profile and the algorithm generates the other successor (node 3).
3. Clearly, the intermediate nodes of the tree represent schedule profiles under construction. When a leaf node that contains a fair PO schedule profile is generated, this solution is stored in $\widehat{s_\Pi}$ and it is used as a bound to prune further branches. Given a node *nd* that represents a partially built schedule profile, we apply an optimistic estimation of the maximum utility that can be obtained from *nd* by

assuming that the expansion of *nd* up to a solution leaf node does not contain empty actions for any agent. Subsequently, the utility of the estimated solution, say $s^{\sim}_{\Pi}$, is compared to the utility of the bound. If $s^{\sim}_{\Pi}$ is unfair or Pareto dominated by the bound, the node *nd* is pruned. Otherwise, *nd* is expanded.

For example, node 5 in Fig. 3 corresponds to a feasible schedule profile $sp1 \in \widehat{s_{\Pi}}$ with associated utilities $u^1 = 9$ and $u^2 = 10$. This allows us to prune the following partially built schedule profiles: (1) node 8 because the schedule profile *sp*2 derived from node 8 is unfair compared to *sp*1; (2) node 9 because the resulting schedule profile *sp*3 is Pareto dominated by *sp*1; and 3) node 11 because the expansion of this node would lead to a schedule profile, *sp*4, as good as *sp*1 (the other schedule profiles *sp*5 and *sp*6 are Pareto dominated by *sp*1).

4. The algorithm returns the solutions of the SG when the search is concluded; in our example, $\widehat{s_{\Pi}} = \{sp1\}$ is the solution found.

The extensive-form algorithm resembles an *alpha-beta* search. On the one hand, a node of the tree represents the move of a player after the moves of its opponents in the preceding levels of the tree. On the other hand, the generation and evaluation of the tree are performed simultaneously and the DFS search ensures that a feasible schedule profile is reached as soon as possible, which will be later used to prune the tree.

The extensive-form algorithm expands first the schedule profiles with fewer empty actions (*monotonicity* property) with the aim to promptly reach a good solution bound. As it occurs in the alpha-beta expansion, the sooner a good bound is reached, the more pruning is applied. On the other hand, note that if the leftmost branch is not pruned, this would represent the ideal schedule of all agents. In short, the DFS expansion together with the chronological backtracking ensures a rational tree expansion, making agents generate first the solutions that report them higher utility (*order* property).

6.4.1 Complexity of the Extensive-Form SG Algorithm

The extensive-form structure is a binary search tree, whose maximal depth is given by the total number of actions of the longest possible schedule profile for the input plan profile Π, which is formally defined as $|s^{-}_{\Pi}| = \sum_{\pi^i \in \Pi} |\pi^i| + \sum_{i \in \mathcal{AG}} |\lambda^i_{\Pi}|$. In other words, each joint action $A_t \in s^{-}_{\Pi}$ includes only one non-empty action for a single agent. We can thus define the complexity of the extensive-form tree algorithm in the worst-case scenario as $\mathcal{O}(2^{|s^{-}_{\Pi}|})$.

In practical terms, a substantial part of the tree is pruned in most cases with the best bound found so far and stored in $\widehat{s_{\Pi}}$, thus reducing the overall complexity of the algorithm.

7 Experimental Results

This section is devoted to experimentally analyze the performance of our FENOCOP framework. Section 7.1 presents a comparative analysis of the two SG algorithms presented in Sect. 6. In Sect. 7.2, we compare the game-theoretic solutions of

FENOCOP (which combines the GG and the SG) with the results of a centralized planner.

7.1 SG Results

The purpose of this test is to evaluate the performance of the two SG algorithms presented in Sect. 6; namely, the *normal-formSG* algorithm and the *ext-formSG* procedure. The tasks of the benchmark comprise one plan π^i per agent forming a plan profile $\Pi = (\pi^1, \ldots, \pi^i, \ldots, \pi^n)$. We used the planner LPG-td (Gerevini and Serina 2002) to generate the individual plan of each agent in Π. All the tasks were run on a single machine[2] with a 30-min timeout.

The benchmark contains tasks of two different planning domains:

- *Transport Domain* This domain is inspired by the well-known *zenotravel* domain of the International Planning Competitions (IPC).[3] Agents are travel agencies that organize their fleets of airplanes to deliver passengers to different destinations. Some of the airplanes are resources shared by the agents; when two or more agents try to use the same plane at the same time, a conflict arises.
- *Space Domain* This is an adaptation of the IPC *rovers* domain. Agents are Mars rovers that navigate through a network of waypoints, analyze samples and communicate the results to a lander, which acts as a communication center. Conflicts arise when agents attempt to analyze the same sample or to simultaneously communicate with the lander.

The main reason for selecting these two domains is that they feature different types of conflicting situations. In the *Transport* domain, conflicts arise when a travel agency uses a plane that another agency is also planning to use. In this case, unless the plane flies back to the original location, the second agency will need to resort to a different plane. On the other hand, conflicts in the *Space* domain arise when two rovers attempt to analyze the same sample simultaneously. In this case, the conflict is solved by making one rover wait until the other one finishes. *Transport* is thus a more constrained domain and, therefore, its solution space is much smaller. In contrast, the *Space* tasks feature many more possible solutions, as well as a significantly larger search space. As we will see in the remainder of this section, the particular characteristics of these two domains will impact the computational results.

The tasks of our benchmark feature 2, 3 or 4 agents. The 2-agent and 3-agent tasks include from 1 to 6 different resources (planes in the *Transport* domain and samples in the *Space* domain). We created four task configurations, each accordingly to a different degree of shared resources among agents and where the highest degree corresponds to the case where all resources are shared and no individual resources are available to the agents. These four variants combined with a maximum

[2] Intel Core i7-3770 CPU at 3.40 GHz, 8 GB RAM.

[3] http://icaps-conference.org/index.php/Main/Competitions.

Table 3 Tasks solved by the *normal-formSG* and the *ext-formSG* algorithms

$\|\mathcal{AG}\|$	Domain	Solvable	*normal-formSG*			*ext-formSG*		
			Solved	Partially	Unsolved	Solved	Partially	Unsolved
2	*Transport*	164	164 (100%)	0	0	164 (100%)	0	0
	Space	240	240 (100%)	0	0	234 (97.5%)	0	6 (2.5%)
3	*Transport*	127	92 (72.4%)	0	35 (27.6%)	127 (100%)	0	0
	Space	236	236 (100%)	0	0	216 (91.6%)	19 (8.0%)	1 (0.4%)
4	*Transport*	53	18 (34.0%)	3 (5.6%)	32 (60.4%)	51 (96.0%)	2 (4.0%)	0
	Space	297	34 (11.4%)	216 (72.7%)	47 (15.9%)	118 (39.7%)	179 (60.3%)	0
Global results		1117	784 (70.2%)	219 (19.6%)	114 (10.2%)	910 (81.5%)	200 (17.9%)	7 (0.6%)

number of 6 resources yield 24 different planning settings. For each planning setting, we generated 10 random tasks with 2 and 3 agents, totalling 240 tasks for each number of agents. In the case of 4-agent tasks, the number of available resources ranges from 1 to 8, which results in 320 tasks. This makes a total of 800 tasks per domain.

In this test, the utility that a plan schedule reports to an agent $i \in \mathcal{AG}$ depends on the *makespan* or finish time of such a schedule. Formally, we define the utility that an agent i obtains from a plan schedule $\psi^i \in \Pi$ as $u^i(\psi^i) = -|\psi^i|$. Therefore, given a schedule profile $s_\Pi = (\psi^1, \ldots, \psi^i, \ldots, \psi^n)$, the utility of an agent i is a value between $-|\pi^i|$ and $-(|\pi^i| + \lambda^i_\Pi)$, depending on the number of empty actions of ψ^i (the schedule of its plan π^i within s_Π).

We first analyze the *coverage* or number of solved tasks by *normal-formSG* and *ext-formSG*, and then, we compare the *computation time* results according to the *size* of the tasks and the number of *conflicts* addressed by the solutions.

7.1.1 Coverage

Table 3 collects the coverage results of both SG algorithms, classified by domain and number of agents. The table shows the percentage of *solved*, *partially solved* and *unsolved* tasks for each setting. A task is said to be *solved* if the search space is exhausted within the 30-minute timeout and so all the task solutions are found. If the time limit expires before the search space is exhausted and at least one solution is returned, we say the algorithm *partially solves* the task. Otherwise, if no solution is found, the task is not solved by the algorithm (*unsolved* task). Note that, for both algorithms, the fairness property is only guaranteed when a task is *solved*. Regarding Pareto optimality, *ext-formSG* guarantees this property when a task is *solved*. In contrast, Pareto optimality is held in any solution obtained by *normal-formSG*.

Before running the tests, we executed all the tasks without a time limit to verify which tasks are *solvable* (see column *Solvable* of Table 3). A task is unsolvable if any of the SG algorithms exhausts the search space without finding a feasible schedule profile. As shown in Table 3, 1117 out of 1600 tasks in the benchmark were found to be solvable. In line with our prior comments on the differences between the two domains, we note that the number of solvable tasks in the *Transport* domain is significantly lower than in the *Space* domain.

We can observe in Table 3 that the SG algorithms are rather sensitive to the number of agents in the game. Both approaches solve most of the tasks with 2 and 3 agents, except for the 3-agent *Transport* setting, where *normal-formSG* fails to solve 27% of the tasks. However, the performance of both methods clearly degrades with 4 agents: *normal-formSG* solves only 34% of the *Transport* tasks and 11% of the *Space* tasks, respectively. On the other hand, *ext-formSG* shows a good performance in the *Transport* domain, but 60% of the *Space* tasks are only *partially solved*.

Table 3 also shows that the *Transport* domain presents the highest number of *unsolved* tasks. Moreover, the *ext-formSG* strategy explores the complete search space in a significantly larger number of *Transport* tasks. As previously mentioned, these results are justified by the more constrained search space of the *Transport* tasks, which comprises fewer solutions compared to the *Space* domain and requires less computation time to be explored.

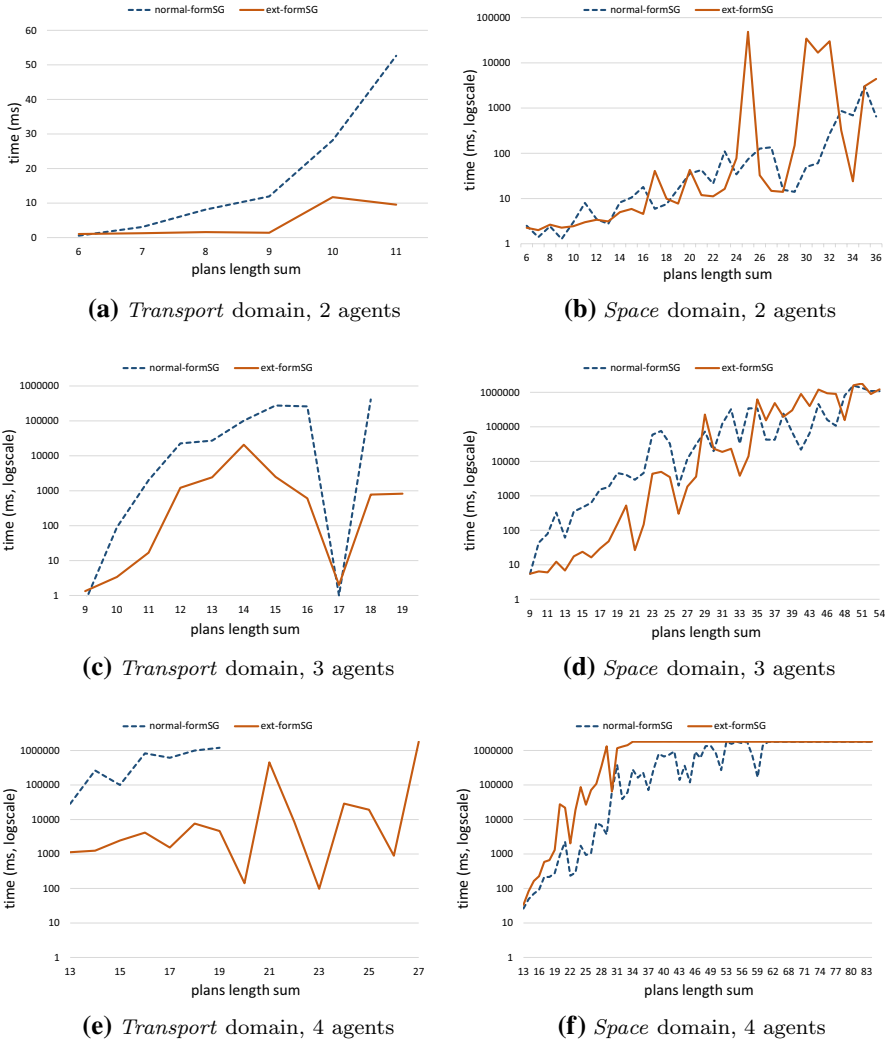
Considering the complexity results of Sects. 6.3.1 and 6.4.1, clearly the 2-agent setting constitutes the sweet spot of the *normal-formSG* approach, because this algorithm explores a tree which is not as deep as the extensive-form tree. Since the branching factor of the *normal-formSG* method is given by the number of agents, $|\mathcal{AG}|$, its performance significantly degrades in the 3-agent tasks and, most notably, with 4 agents, where it is clearly outperformed by the *ext-formSG* approach. Particularly, it only solves 34% of the *Transport* tasks and 11% of the *Space* instances. Yet, *normal-formSG* obtains in general good results, solving almost 70% of the tasks and partially attaining 20% of the instances.

In contrast, the *ext-formSG* approach is proven to scale up much better due to the reduced branching factor of the tree and the effectiveness of its pruning mechanisms. In fact, this approach solves (completely or partially) more than 99% of the benchmark (910 tasks *solved* and 200 tasks *partially solved*, out of 1117 *solvable* instances), only failing to generate a solution for 7 tasks.

The difference in the coverage results of both approaches is also explained by the fact that *normal-formSG* does not leverage the appearance of the same conflict in similar schedule profiles. The *ext-formSG* algorithm exploits better this situation because it generates the schedule profiles incrementally. Hence, once a conflict is detected, the infeasible branch is directly discarded, thus saving a significant amount of computation time.

7.1.2 Computation Time with Respect to Task Size

Figure 4 compares the computation time required by both SG algorithms with respect to the size of the tasks included in our benchmark. Each data point in the plots represents the average computation time (in milliseconds) for tasks whose



(a) *Transport* domain, 2 agents

(b) *Space* domain, 2 agents

(c) *Transport* domain, 3 agents

(d) *Space* domain, 3 agents

(e) *Transport* domain, 4 agents

(f) *Space* domain, 4 agents

Fig. 4 Computation time results of the *normal-formSG* and the *ext-formSG* algorithms according to task size

input plan profiles have the same number of actions. For instance, a value of 10 on the horizontal axis represents the plan profiles that contain a total of 10 actions. Note that a data point of 1,800,000 ms in Fig. 4 (30 min) indicates that the timeout was reached in the execution of all the corresponding plan profiles; in other words, these tasks were *partially solved*. Finally, in case no task of a given size is solved by either *normal-formSG* or *ext-formSG*, the data point is not plotted. For example, Fig. 4c shows that *normal-formSG* does not solve any of the tasks with an associated plan profile of 19 or more actions.

In general, the size of the plan profiles in the *Transport* domain is significantly lower (up to 11, 19 and 27 actions, depending on the number of agents) than the *Space* domain (with a maximum of 36, 54 and 83 actions, respectively). None of the two SG algorithms was able to solve the largest instances of the *Transport* domain because of the high number of conflicts among agents.

In the *Transport* domain, the *ext-formSG* clearly outperforms the *normal-formSG*, as shown in Fig. 4a, c, e. This fact becomes more evident in large instances that involve 3 and 4 agents. As noted above, the *normal-formSG* algorithm is particularly sensitive to the number of agents because this parameter determines the branching factor of the search tree. The performance of the *ext-formSG* algorithm, however, is not so dependent on the number of agents because this algorithm always builds a binary tree. *ext-formSG* solves most of the instances of the *Transport* domain in less than a minute, and scales up significantly better than *normal-formSG*.

Regarding the *Space* domain, the *normal-formSG* approach presents a more steady behavior (see Fig. 4b, d, f). In this domain, *ext-formSG* is significantly more expensive to reach a convergence point in some of the tasks. This is because, in the *Space* domain, *ext-formSG* is unable to effectively prune the tree due to the relative lack of conflicts, which results in a large search space to explore. Moreover, *ext-formSG* partially solves a large number of tasks (particularly in the 4-agent setting), which explains the 30-min values in Fig. 4f. In contrast, the computation times of the *normal-formSG* approach are significantly lower because the order property of the SG is better exploited in this approach. This, together with the low number of conflicts of the *Space* domain, makes the *normal-formSG* a very efficient approach to find feasible schedule profiles in this domain.

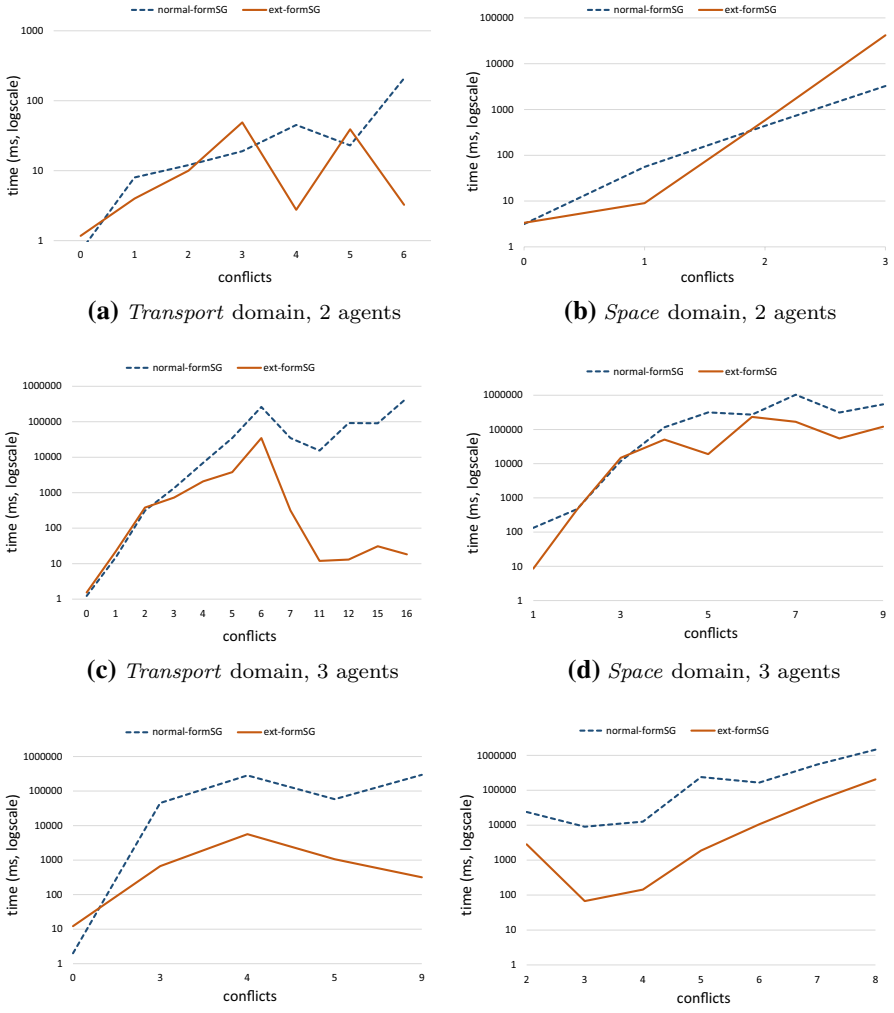
All in all, although the *normal-formSG* method slightly outperforms the *ext-formSG* algorithm in domains that have a low number of conflicts, the *ext-formSG* algorithm presents in general a more consistent behavior and it is particularly suitable for tasks with a large number of conflicts.

7.1.3 Computation Time with Respect to Task Conflicts

In this section, we analyze the influence of the conflicts in the computation time of both SG algorithms. We evaluate the number of conflicts that arise in a task as the total loss of utility of the solution; that is, we measure the number of conflicts as the number of empty actions that need to be introduced in a schedule profile to solve the arising conflicts. In other words, given the schedule profile s_{Π} computed by an SG algorithm for an input plan profile Π, the number of conflicts of the task is calculated as $|s_{\Pi}| - |\Pi|$.

Figure 5 depicts the results of this experiment considering only the tasks *solved* by both SG algorithms, ignoring the *partially solved* tasks. Each data point shown in the plots represents the average computation time (in milliseconds) of the tasks whose solutions (schedule profiles) have the same number of empty actions (horizontal axis).

In general, we can observe that the computation time increases with the number of conflicts in both SG algorithms. It is also noticeable some peak values in tasks with a relatively low number of conflicts (see Fig. 5c). This usually happens



(a) *Transport* domain, 2 agents

(b) *Space* domain, 2 agents

(c) *Transport* domain, 3 agents

(d) *Space* domain, 3 agents

(e) *Transport* domain, 4 agents

(f) *Space* domain, 4 agents

Fig. 5 Computation time results of the *normal-formSG* and the *ext-formSG* algorithms according to task conflicts

in large tasks because both SG algorithms are sensitive to the task size, as discussed in the previous section. Generally speaking, the *Space* domain features fewer conflicts than the *Transport* domain.

The computational time tends to increase in the *Transport* domain with the number of conflicts, particularly in the *normal-formSG* algorithm, whose performance is clearly compromised as the number of conflicts raises (see Fig. 5a, c). On the other hand, *ext-formSG* exhibits a more stable performance in this domain, showing shorter computation times in general. All in all, *ext-formSG* is clearly

more efficient than *normal-formSG* at solving complex *Transport* tasks featuring a large number of conflicts.

Regarding the *Space* domain, *ext-formSG* outperforms *normal-formSG* in most 2-agent tasks, but it presents a higher average computation time for tasks with 3 conflicts (see Fig. 5b). In the case of 3-agent tasks, the computation time raises exponentially as the number of conflicts increases. Again, *ext-formSG* outperforms *normal-formSG* in most 3-agent tasks (see Fig. 5d). Most of the 4-agent *Space* tasks were *partially solved* by both approaches, or *unsolved* by *normal-formSG* (see Table 3), and so they were excluded from Fig. 5f. In general, *ext-formSG* has significantly lower computation times than *normal-formSG* in this setting.

The results confirm that *ext-formSG* is better suited to deal with conflicts since it explores a binary search tree and prunes the branches with conflicts. In summary, *ext-formSG* applies a *DFS* strategy in a tree depth-bounded by $\sum_{\forall i \in \mathcal{AG}} \lambda_{\Pi}^{i}$ and it is able to find a first feasible outcome early on (even in tasks that involve a large amount of conflicts). This outcome as well as further refinements of this solution will be used to efficiently prune the search tree.

On the other hand, the *normal-formSG* algorithm follows a *BFS* strategy where every node is a schedule profile. Nodes are introduced from the highest to the lowest quality (the deeper the node is, the more empty actions it has) and the pruning mechanism discards any successor of a feasible PO schedule profile by Pareto dominance. In domains that feature multiple conflicts, such as the *Transport* and *Space* domains, the BFS will take time to find a solution, possibly at a very deep level of the tree if the number of conflicts is relatively high. This limits the pruning effectiveness, resulting in a very costly process, which justifies the poor results of this SG algorithm. Moreover, the branching factor increases with the number of agents, which compromises the overall performance of this method, as shown in Fig. 5e, f

All in all, the results obtained in this experiment prove that *ext-formSG* is more tolerant to conflicts than *normal-formSG*. These results are in line with the theoretical analysis presented in Sects. 6.3.1 and 6.4.1. Furthermore, unlike *normal-formSG*, *ext-formSG* is able to fully or partially solve most of the tasks in the benchmark, as shown in Table 3. We can thus conclude that *ext-formSG* is more stable than *normal-formSG* in complex tasks with many conflicts, thanks to its robust search strategy and its efficient pruning mechanism.

7.2 FENOCOP Results

In this section, we perform an evaluation of the global planning task when agents have several plans in their sets Γ^i, which requires executing both the GG and the SG of FENOCOP. Particularly, we want to compare the quality of the FENOCOP solutions against the solutions of the centralized planner LPG-td (Gerevini and Serina 2002) with respect to Pareto optimality and fairness. We must first point out a couple of observations:

1. The PO and fair equilibrium schedule profiles returned by the *normal-formSG* or *ext-formSG* algorithms for each plan profile are stored in the utility matrix of the

GG (see illustrative example in Fig. 1). Then, we solve the normal-form GG and we return a solution that is a Nash equilibrium. However, this does not guarantee that the solution returned by the GG is PO and fair, since, in contrast to the SG, it does not guarantee these two properties. Consequently, we must check which of the NE solutions returned by the GG satisfy the PO and fairness solution concepts.

2. In LPG-td, we used the standard objective function that minimizes the number of actions so the planner returns the best possible global solution with respect to this optimization criterion. Since a centralized planner does not individually reason on the plan of each agent but on the global plan as a whole, solutions may exhibit a non-equitable distribution of the utilities. This is precisely the key point of comparison between the FENOCOP and LPG-td solutions.

FENOCOP is the result of integrating a variety of tools, including existing technologies and explicitly designed resources. The input parameters of FENOCOP are the planning tasks of the agents, which are encoded as STRIPS tasks (Fikes and Nilsson 1971). We run LPG-td for solving each planning task and the solution plans are stored in the set Γ^i of each agent. For each combination of plans or plan profile Π, an instance of the SG is launched and solved with one of the SG algorithms presented in Sect. 6. Once the utilities of the SG solutions are saved in the normal-form matrix of the GG, the NE solutions of the GG are computed by means of the Gambit tool (McKelvey et al. 2014). Finally, from the set of stable solutions, if there are more than one, we select a PO and fair outcome as the final solution of the task.

This benchmark includes 20 3-agent tasks from the *Transport* domain. *Transport* was chosen for this test as it is the most challenging domain in our SG benchmark, giving rise to complex instances with a wide variety of conflicts. For FENOCOP, a benchmark task was encoded as an independent STRIPS task per agent and each task was solved with LPG-td. For running LPG-td as a centralized planner, we encoded each benchmark task as a single global planning task. We used the standard objective function of minimizing the number of actions in both configurations of LPG-td.

Table 4 collects the experimental results of this test. Columns labeled with u^i show the utility of the three agents in both configurations. Note that, as in the SG test, the utility that a plan schedule ψ^i reports to an agent i is defined as $u^i(\psi^i) = -|\psi^i|$. *PO* and *fair* columns indicate respectively, whether or not the plan obtained by an approach is Pareto optimal and fair with respect to the plan yielded by the other method. In other words, we compare the solution of LPG-td against that of FENOCOP in order to assess which one is PO and fair.

The results clearly prove that FENOCOP is the superior approach when it comes to attain a feasible solution that is also efficient (PO) and represents a balance of the agents satisfaction (equitable loss of utility). As shown in Table 4, all the FENOCOP solutions are PO and all but one are fair. Note that the solution of *tra*4 is unfair because the least satisfied agent (agent 2 with $u^2 = -6$) would be more satisfied with the utility of the LPG-td solution ($u^2 = -5$).

Table 4 Solution plans obtained by FENOCOP and the centralized planner LPG-td

	LPG-td					FENOCOP				
	u^1	u^2	u^3	PO	Fair	u^1	u^2	u^3	PO	Fair
tra1	−8	−4	−5	✗	✗	**−5**	**−3**	−5	✓	✓
tra2	−5	−4	−5	✓	✓	−5	−4	−5	✓	✓
tra3	**−3**	−4	−6	✓	✗	−4	−4	**−4**	✓	✓
tra4	−4	**−5**	−4	✓	✓	−4	−6	**−3**	✓	✗
tra5	−4	−4	−3	✓	✓	−4	−4	−3	✓	✓
tra6	−3	−4	−3	✓	✓	−3	−4	−3	✓	✓
tra7	−5	−4	−3	✓	✓	−5	−4	−3	✓	✓
tra8	−3	−3	−5	✗	✗	−3	−3	**−4**	✓	✓
tra9	−3	−5	−5	✗	✗	−3	**−3**	−5	✓	✓
tra10	−5	−3	−6	✗	✗	**−4**	−3	**−5**	✓	✓
tra11	−4	−5	−5	✗	✗	**−3**	**−4**	−5	✓	✓
tra12	−3	−3	−4	✓	✓	−3	−3	−4	✓	✓
tra13	−3	−5	−3	✓	✓	−3	−5	−3	✓	✓
tra14	−4	−3	−4	✓	✓	−4	−3	−4	✓	✓
tra15	−5	−3	−4	✗	✗	**−4**	−3	**−3**	✓	✓
tra16	−4	−4	−4	✗	✗	−4	**−3**	−4	✓	✓
tra17	−4	−4	−4	✗	✗	−4	**−3**	−4	✓	✓
tra18	−10	−6	−12	✗	✗	**−7**	−6	**−4**	✓	✓
tra19	−4	−4	−4	✗	✗	−4	**−3**	−4	✓	✓
tra20	−5	−3	−7	✗	✗	**−4**	−3	**−5**	✓	✓
Global				45%	40%				100%	95%

Utilities of specific agents that outperform their counterpart between the two approaches are shown in bold

As expected, less than half of the solutions of LPG-td when run as a centralized planner are PO and fair. Since the planner decisions are based on global optimization and non-individualized reason, most of the outcomes are non-efficient and unfair. This is interpreted as a solution that reports a low degree of satisfaction to the involved parties.

Another key advantage of FENOCOP is that, as opposite to centralized planners, we ensure the generation of a stable (Nash equilibrium) solution. Let us illustrate this through a 2-agent example task based on the *Transport* domain. The task features two travel agencies (agents) that must organize each a trip for a passenger (`p1` and `p2`, respectively). Passenger `p1` starts at city `c1` and wants to travel to `c2`, while `p2` is at `c2` and wants to visit `c4`. Aircraft `a1` is located at `c3` and `a2` is at `c2`.

FENOCOP generates two individual plans per agent, as shown in Table 5 (π_1^1 and π_2^1 for agent 1, and π_1^2 and π_2^2 for agent 2). The inherent utilities of these plans are also displayed in Table 5. The SG is invoked with the four possible combinations of plans ($\Pi_{11} = (\pi_1^1, \pi_1^2)$, $\Pi_{12} = (\pi_1^1, \pi_2^2)$, $\Pi_{21} = (\pi_2^1, \pi_1^2)$ and $\Pi_{22} = (\pi_2^1, \pi_2^2)$) in order to generate four feasible schedule profiles that are stable, PO and fair.

Table 5 Individual agents' plans synthesized by FENOCOP for the example *Transport* task

Time	π_1^1	π_2^1	π_1^2	π_2^2
0	`fly a2 c2 c1`	`fly a1 c3 c2`	`board p2 a2 c2`	`fly a1 c3 c2`
1	`board p1 a2 c1`	`fly a1 c2 c1`	`fly a2 c2 c1`	`board p2 a1 c2`
2	`fly a2 c1 c2`	`board p1 a1 c1`	`fly a2 c1 c4`	`fly a1 c2 c3`
3	`debark p1 a2 c2`	`fly a1 c1 c2`	`debark p2 a2 c4`	`fly a1 c3 c4`
4		`debark p1 a1 c2`		`debark p2 a1 c4`
Utilities	$u^1(\pi_1^1) = -4$	$u^1(\pi_2^1) = -5$	$u^2(\pi_1^2) = -4$	$u^2(\pi_2^2) = -5$

Table 6 GG utility matrix of the example *Transport* task

	π_1^2	π_2^2
π_1^1	$-7, -4$	$-4, -5$
π_2^1	$\mathbf{-5, -4}$	$-\infty, -\infty$

NE outcome in bold

The utilities of the resulting feasible schedule profiles are reflected in the GG utility matrix of Table 6, which shows that the solution chosen by FENOCOP is the schedule profile that combines π_2^1 and π_1^2, with associated utilities $u^1 = -5$ and $u^2 = -4$. Despite having the same sum of utilities as the schedule profile that combines π_1^1 and π_2^2 (-9 units), the solution chosen by FENOCOP is the only NE of Table 6, since no agent would benefit from unilaterally deviating from this strategy.

We solved this example task with LPG-td, and the returned solution was π_1^1, π_2^2 (see Table 6). As previously stated, a centralized planner, such as LPG-td, optimizes a global metric (in this case, the number of actions of the plan). For this reason, LPG-td considers the solution that combines π_2^1 and π_1^2 as good as the combination of π_1^1 and π_2^2, since they equally minimize the objective function of LPG-td. Hence, LPG-td returns any of these solutions indistinctly. Since the LPG-td solution for this example task is not a NE, the agents could deviate from this outcome and execute their alternative plans instead, which could potentially cause conflicts that would endanger the executability of the plans.

We can thus conclude that FENOCOP is an appropriate approach to solve the non-cooperative planning task formulated in this paper. Our approach not only guarantees the selection of a Nash equilibrium solution for the task, but also guarantees that the solutions hold the Pareto optimality and fairness properties in most cases, as empirically demonstrated. In contrast, centralized planners focus on the optimization of global magnitudes, failing in most cases to attain a stable, PO and fair solution that properly balances the utilities of the self-interested agents.

8 Conclusions

In this paper, we presented FENOCOP, a game-theoretic approach for non-cooperative agents that want to execute their plans in a shared environment. Each agent generates a collection of plans that attain its individual task, and takes part on a game that allows the participants to jointly select a feasible schedule profile that guarantees the concurrent execution of their plans. FENOCOP includes two different games: the General Game (GG) is a general-sum game in which agents select the schedule profile to execute among the set of executable combinations of their plans. The generation of a feasible schedule profile for each combination of the agents' plans is taken care of by means of the Scheduling Game (SG). In the SG, agents study how to schedule their individual plans in order to ensure their executability. Agents address conflicts by delaying the execution of their actions while trying to maximize their utilities.

Whereas the solutions of the GG are guaranteed to be Nash Equilibria (NE), the outcomes of the SG hold two additional solution concepts: *Pareto optimality* allows to return the best outcome among the stable feasible schedule profiles of an SG, and *fairness* maximizes the utility of the least satisfied agent. The satisfaction of these concepts maximizes the quality of the schedule profiles among which the GG selects the solution of the planning problem.

We introduced and experimentally validated two algorithms that address the SG problem. We also implemented the overall FENOCOP framework by combining the GG together with the SG. The NE solution of the GG is obtained by means of the Gambit tool (McKelvey et al. 2014).

The results of the SG algorithms reveal that the *ext-formSG* approach scales up significantly better than the *normal-formSG* approach, thanks to its steady branching factor and the effectiveness of the pruning mechanisms applied. In contrast, the *normal-formSG* algorithm performs slightly better than *ext-formSG* in small instances that involve a reduced amount of conflicts among agents.

Regarding the FENOCOP results, we confirmed that, as expected, our approach is more effective than a centralized planner at satisfying the agents' interests and fairly balancing their utilities. In contrast, centralized planners focus on optimizing some metric of the planning task as a whole, being unaware of the presence of self-interested agents with independent objectives.

In conclusion, we defined and empirically validated an approach that realistically addresses the non-cooperative multi-agent planning problem. FENOCOP leverages the utilities of self-interested agents and promotes the individual satisfaction of the participants through a set of algorithms which aim for the generation of solutions that are stable, Pareto optimal and fair.

FENOCOP follows a two-level game scheme which separates the problem of selecting an executable plan combination from the generation of the executable schedule profiles. This division aims to reduce the computational complexity of the task. However, one can observe that, even with this decomposition, the task we aim to solve has exponential complexity, which limits the applicability of FENOCOP to problems of a relatively restrained size.

For this reason, as a future work, we intend to focus on the synthesis of individual plans. FENOCOP assumes that each agent has a pre-calculated collection of plans and schedules their actions until all the combinations of agents' plans fit together. We believe that the complexity of this costly problem could be significantly alleviated if we equip each agent with the appropriate resources to synthesize a response that directly fits with the rest of agents' plans. Our goal, therefore, is to study the state of the art in multi-agent planning in order to come up with a planning technique that allows an agent to synthesize plans that can be directly integrated with the rest of agents' proposals, thus overcoming the costly task of combining pre-existing individual plans via the delay of individual actions.

Acknowledgements This work is supported by the Spanish MINECO project TIN2017-88476-C2-1-R. Jaume Jordán is funded by grant APOSTD/2018/010 of Generalitat Valenciana - Fondo Social Europeo and by UPV PAID-06-18 project.

References

Blum A, Furst ML (1997) Fast planning through planning graph analysis. Artif Intell 90(1–2):281–300

Bowling MH, Jensen RM, Veloso MM (2003) A formalization of equilibria for multiagent planning. In: Proceedings of the 18th international joint conference on artificial intelligence (IJCAI), pp 1460–1462

Brafman RI, Domshlak C, Engel Y, Tennenholtz M. Planning games. In: Proceedings of the 21st international joint conference on artificial intelligence (IJCAI), pp 73–78 (2009)

Brandt F, Conitzer V, Endriss U, Lang J, Procaccia AD (eds) (2016) Handbook of computational social choice. Cambridge University Press, Cambridge

Buzing P, Mors AT, Valk J, Witteveen C (2006) Coordinating self-interested planning agents. Auton Agents Multi-Agent Syst 12(2):199–218. https://doi.org/10.1007/s10458-005-6104-4

Chevaleyre Y, Dunne PE, Endriss U, Lang J, LemaÎtre M, Maudet N, Padget J, Phelps S, Rodríguez-Aguilar JA, Sousa P (2006) Issues in multiagent resource allocation. Informatica 30(1):3–31

Crosby M, Rovatsos M (2011) Heuristic multiagent planning with self-interested agents. In: Proceedings of the 10th international conference on autonomous agents and multiagent systems (AAMAS), vol 1–3, pp 1213–1214

Endriss U, Maudet N, Sadri F, Toni F (2006) Others: negotiating socially optimal allocations of resources. J Artif Intell Res 25:315–348

Fikes R, Nilsson N (1971) STRIPS: A new approach to the application of theorem proving to problem solving. Artif Intell 2(3):189–208

Gal K, Procaccia AD, Mash M, Zick Y (2018) Which is the fairest (rent division) of them all? Commun ACM 61(2):93–100

Gerevini A, Serina I (2002) Lpg: a planner based on local search for planning graphs with action costs. In: Proceedings of the 6th international conference on artificial intelligence planning systems. AAAI Press, pp 13–22. http://dl.acm.org/citation.cfm?Id=3036884.3036887

Ghallab M, Nau D, Traverso P (2004) Automated planning: theory & practice. Elsevier, Amsterdam

Gillies DB (1959) Solutions to general non-zero-sum games. Contrib Theory Games 4(40):47–85

Jensen RM, Veloso MM, Bowling MH (2001) Obdd-based optimistic and strong cyclic adversarial planning. In: Proceedings of the 6th European conference on planning, pp 265–276

Jordán J, Onaindía E (2015) Game-theoretic approach for non-cooperative planning. In: Proceedings of the 29th AAAI conference on artificial intelligence (AAAI), pp 1357–1363

Komenda A, Stolba M, Kovacs DL (2016) The international competition of distributed and multiagent planners (CoDMAP). AI Mag 37(3):109–115

Larbi RB, Konieczny S, Marquis P (2007) Extending classical planning to the multi-agent case: a game-theoretic approach. In: Symbolic and quantitative approaches to reasoning With uncertainty, 9th European conference, ECSQARU, pp 731–742

McKelvey RD, McLennan AM, Turocy TL (2014) Gambit: software tools for game theory, version 13.1.2. http://www.gambit-project.org. Accessed 2 Oct 2015

Myerson RB (1981) Utilitarianism, egalitarianism, and the timing effect in social choice problems. Econometrica 49(4):883–897

Myerson RB (2013) Game theory. Harvard University Press, Cambridge

Nguyen N, Katarzyniak R (2009) Actions and social interactions in multi-agent systems. Knowl Inf Syst 18(2):133–136

Nissim R, Brafman R I (2013) Cost-optimal planning by self-interested agents. In: Proceedings of the 27th AAAI conference on artificial intelligence, pp 732–738. http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6384

Osborne MJ, Rubinstein A (1994) A course in game theory. MIT Press, Cambridge

Rawls J (1971) A theory of justice. Harvard University Press, Harvard Paperback

Sailer F, Buro M, Lanctot M (2007) Adversarial planning through strategy simulation. In: 2007 IEEE symposium on computational intelligence and games, pp 80–87. https://doi.org/10.1109/CIG.2007.368082

Shoham Y, Leyton-Brown K (2009) Multiagent systems: algorithmic, game-theoretic, and logical foundations. Cambridge University Press, Cambridge

Torreño A, Onaindia E, Komenda A, Stolba M (2018) Cooperative multi-agent planning: a survey. ACM Comput Surv 50(6):84:1–84:32

Torreño A, Onaindia E, Sapena Ó (2014) FMAP: distributed cooperative multi-agent planning. Appl Intell 41(2):606–626

Van Der Krogt R, De Weerdt M (2005) Self-interested planning agents using plan repair. In: ICAPS 2005 workshop on multiagent planning and scheduling, pp 36–44

Von Neumann J, Morgenstern O (2007) Theory of games and economic behavior. Princeton University Press, Princeton

Weerdt MD, Clement B (2009) Introduction to planning in multiagent systems. Multiagent Grid Syst 5(4):345–355