Adaptive Human-Robot Collaboration: Evolutionary Learning of Action Costs Using an Action Outcome Simulator

Silvia Izquierdo-Badiola[1,2], Guillem Alenyà[2] and Carlos Rizzo[1]

***Abstract*— One of the main challenges for successful human-robot collaborative applications lies in adapting the plan to the human agent's changing state and preferences. A promising solution is to bridge the gap between agent modelling and AI task planning, which can be done by integrating the agent state as action costs in the task planning domain. This allows for the plan to be adapted to different partners, by influencing the action allocation. The difficulty then lies in setting appropriate action costs. This paper presents a novel framework to learn a set of planning action costs considering the preferred actions for an agent based on their state. An evolutionary optimisation algorithm is used for this purpose, and an action outcome simulator is developed to act as the black-box function, based on both an agent model and an action type model. This addresses the challenge of collecting data in HRC real-world scenarios, accelerating the learning for posterior fine-tuning in real applications. The coherence of the models and the simulator is proven through a conducted survey, and the learning algorithm is shown to learn appropriate action costs, producing plans that satisfy both the agents' preferences and the prioritised plan requisites. The resulting system is a generic learning framework integrating components that can be easily extended to a wide range of applications, models and planning formalisms.**

I. Introduction

The topic of Human-Robot Collaboration (HRC) has gained substantial attention over the past years, with the idea of combining strengths from humans and robots to improve efficiency and productivity in shared plans. However, several challenges still need to be tackled when developing such systems for practical real-world applications, such as dealing with the variable human state, preferences and capabilities.

AI (or automated) task planning frameworks have been implemented to generate and distribute the sequence of actions required to achieve a shared goal, when given a world model (domain) with an initial state and a high-level goal (problem). In a collaborative plan, the assignment of actions to different agents should be based on the states and preferences of the contributing members. In our previous work [1], we targeted the gap between agent state modelling and AI task planning, by developing a planning framework for HRC plans integrating the agents' states into the action costs in the planning domain. This would influence the action allocation between the agents based on their state, which was defined in terms of three elements: *capacity*, *motivation* and *knowledge*. The action costs reflecting the agent model elements must be given appropriate values, as these will have a direct impact on the generated plan, and therefore on its successful execution by the contributing agents. Manually estimating the costs is a difficult task, and can lead to unexpected results. Learning from experience constitutes a valid approach for this purpose [2], [3], but collecting training data in HRC applications is a well-known challenge. This creates a need for the development of a human action simulator, which is in turn hindered by the lack of appropriate models. In addition, we recognise that learning over a collection of domains with different structures constitutes a major challenge in learning for AI task planning. We propose to address these challenges by developing the novel framework shown in Fig. 1, with the objective of facilitating and accelerating the learning of action costs associated with an agent type, using a simulator in a black-box optimisation algorithm. The framework includes the following elements, constituting the contributions of this paper:

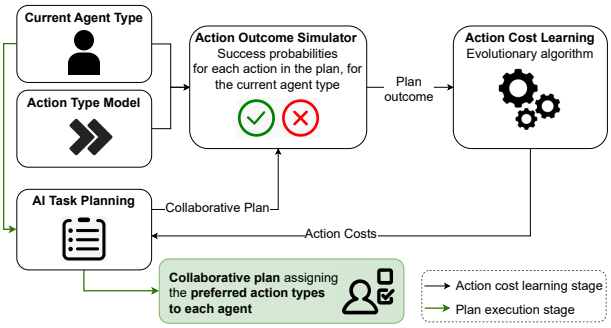

Fig. 1. Framework developed to learn the action costs corresponding to each agent in a human-robot collaboration. The framework implements an evolutionary algorithm for learning, where the black-box function consists of an action outcome simulator based on an agent and an action type model, all designed in this work.

- **Agent State Model** - We define an agent model inspired by character attributes in role-playing games (RPG) adapted to human-robot collaborative tasks (Sec. III), which has not been explored in literature before.
- **Action Type Model** - Generally, the agent model elements are assumed to impact the outcome of all actions in the same way. We recognise that this is not the case, and define what we call an action type model, grouping actions in HRC scenarios into a number of

[1]Eurecat, Centre Tecnològic de Catalunya, Robotics and Automation Unit, Barcelona, Spain {silvia.izquierdo, carlos.rizzo}@eurecat.org

[2]Institut de Robòtica i Informàtica Industrial, CSIC-UPC, Barcelona, Spain {sizquierdo,galenya}@iri.upc.edu

*S. Izquierdo is a fellow of Eurecat's *Vicente López* PhD grant program. This work has been partially supported by MCIN/ AEI /10.13039/501100011033 under the project CHLOE-GRAPH (PID2020-119244GB-I00); by MCIN/ AEI /10.13039/501100011033 and by the "European Union (EU) NextGenerationEU/PRTR under the project ROB-IN (PLEC2021-007859).

action types based on how much each element in the agent state model affects the successful execution of the action (see Sec. IV).
- **Action Outcome Simulator** - Motivated by the huge challenge of obtaining real-world data in this type of scenario, an action outcome simulator is devised (see Sec. V), providing the outcome of the actions (success or failure) in a plan based on the agent state and the defined action type model. This simulator is used as the black-box function in the learning component.
- **Action Cost Learning** - Addressing the difficulty of learning action costs in AI task planning, we present a general method implementing a black-box optimisation algorithm to learn appropriate costs, regardless of the planner and domain structure (see Sec. VI).

The integration of these elements into a framework provides a fast and effortless way of learning costs that produce adaptive collaborative plans, considering the preferred actions for an agent type. Although the action outcome simulator is not intended to match a real application, it constitutes a way of speeding up the initial learning, for future fine-tuning in a real scenario. The validity of the simulator is backed up by a user survey in Sec.VII-A, and some examples of learning appropriate action costs with the proposed method are presented in Sec.VII-B.

II. Related Work

Human modelling for task planning in HRC. For successful human-robot collaborative plans, the task planning framework should assign the appropriate complementary actions to the team members based on their state and capabilities. In our previous work [1], we targeted the gap between human modelling and its effective integration in the planning framework. This is something other works have attempted to do [4], [5], with some of the human elements modelled including mental state factors such as knowledge [6], [7], capacity, distraction and fatigue [8]–[10]. Cooper [11] defined the concept of Persona as a fictional user model that represents archetypical users. This model is intended to focus only on the attributes that are relevant to the specific task and context, and has been applied to HRI by Andriella *et al.* [12]. This inspired us to look into the field of RPG [13]–[15], where substantial effort has been made towards developing models for character definition. Based on our literature review, these models have not been translated into HRC robotic applications yet. We believe in the potential of adapting these models for modelling humans and creating agent types in HRC scenarios, which is a novelty proposed in this work. On another note, we realise that none of these works attempt to model the link between the elements in the human model and their different effect on each action in the plan. We refer to this as an action type model, and integrate it into the framework along with the agent model.

Learning in AI task planning. Learning algorithms have been used to assist AI planning, by learning action models and control knowledge to guide the search [16]. In [17]–[19], evolutionary genetic algorithms are used to guide AI planning. Learning action costs for symbolic planning has been explored in a number of works [2], [3], [20], where the costs are updated from sensed data during plan execution using methods such as relational decision trees or an exponentially weighted moving average. The problem of having to estimate the costs at episode 0 without previous knowledge is mentioned as unsolved. In the assistive scenario, Canal *et al.* [21] focus on learning to improve the robot's behaviour and adapt it to the user preferences. They achieve this by modifying the actions' costs associated with a user model in task planning, learning from user feedback at the end of the execution. The challenges of data collection and learning over different planning domains remain present, and there does not seem to be significant research on learning action costs specified in the domain for automated planning. We therefore explore the possibility of learning these by using a simulator and a black-box learning framework that can be deployed for any planner and domain.

Simulating human behaviour. The challenge of collecting training data in real applications is tackled by Andriella *et al.* in [12], where the authors develop a Persona simulator to learn a robot's initial policy in a cognitive training scenario. We translate this concept for action cost learning in HRC plans using a number of simulated agent types. The lack of realistic synthetic agents to simulate human behaviour is recognised as one of the main obstacles to developing effective HRC applications [22], and is targeted by a number of works [23]–[26]. The synthetic human agents are far from being natural and realistic, and achieving this would require really complex systems to be developed. Without intending to match this level of realism, we recognise the potential of developing simple systems targeting only the required simulated elements (e.g. success or failure based on the agent state). This accelerates the initial evaluation of HRC applications, before fine-tuning the case in the real world.

III. Agent State Model and Agent Types

For collaborative plans to be successful, the agents' states need to be continuously monitored, so that the plan can be adapted to any change in them. An agent state model definition is required and should cover enough elements representing what the agents' preferred actions would be towards the completion of the shared goal. An agent state model AS comprising m elements E can be defined as

$$AS = \{E_1, ..., E_m\}. \tag{1}$$

The agent model we define in this work is an extension of the one presented in [1] and is based on literature research adapted to HRC scenarios, as described below. The model includes physical and mental factors, with three static elements (Knowledge (K), Dexterity (D) and Strength (S)), and two dynamic elements (Focus (F) and Vitality (V)), which might vary during the plan execution. Table I provides a definition of the elements in the agent state model $AS = \{K, D, S, F, V\}$. The selection of these elements has been made by targeting collaborative tasks examples in both industrial and household applications, such as tidying up

TABLE I

DEFINITION OF THE ELEMENTS IN THE AGENT STATE MODEL

	Element	Definition
Static	**Knowledge (K)**	Prior knowledge of the agent on the task.
	Dexterity (D)	Agility, coordination and skill in performing tasks with the hands.
	Strength (S)	Physical power and carrying capacity.
Dynamic	**Focus (F)**	Level of concentration during the execution of the task. Opposite of distraction.
	Vitality (V)	Levels of energy and activity during the execution of the task. Opposite of fatigue.

a workshop or cooking. The definition of the three static elements has mostly been inspired by character attributes in RPG [13]–[15], representing the character's natural abilities and aptitudes. We have taken the most relevant elements for our target application: strength and dexterity as physical abilities, and knowledge (encompassing intelligence and wisdom), as a mental ability. The two dynamic elements have been chosen based on literature [9], [10], [27], [28], where the concepts of distraction (versus focus) and fatigue (versus vitality) have been used, mostly related to the agent contribution level towards the goal and to the safety related to this contribution. These five elements have a direct effect on the successful execution of most actions in a collaborative task, and imply that an agent might prefer or perform better in certain types of actions in the plan over other types.

From this model, we can define a set of agent types that can be recognised during the plan execution by sensing the current level of each element. Each element is measured in a discretised way, accepting the values of *bad*, *fair*, or *good* (0.1, 0.5, or 1). This leads to 243 (3^5) agent types, representing a reasonable number of types to cover the possible agent states during a collaboration. Once these agent types have been defined, their corresponding action costs can be learnt. Observe that a single agent can be matched to different agent types during the collaboration based on their state. Whenever a change is sensed, a replan is triggered using the action costs associated with the new agent type, which might lead to an alternative task allocation.

IV. ACTION TYPE MODEL

We recognise that the successful outcome of each action is affected differently by the different elements in the agent model. As an example, grasping a small and pointy object requires more dexterity and focus than pushing a heavy box, which requires more strength and vitality. The impact that each element in the agent state model (AS) has on the success of an action has been embodied into an action type model (AM), containing a number n of action types (AT):

$$AM = \{AT_1, ..., AT_n\}. \tag{2}$$

Each action type *AT* is defined by a set of impact weights w_E describing how much each element E in AS influences its successful execution as

$$AT = \{w_{E_1}, ..., w_{E_m}\} \tag{3}$$

where m is the number of elements in the agent model. Each weight can take the values of *low (L), medium (M), high (H)*, representing a low, medium or high impact on the successful action outcome. Applying this to the agent model defined in Sec. III, an action type is defined as $AT = \{w_K, w_D, w_S, w_F, w_V\}$. When assigning values to the weights, the sum of the five weights must be equal to 1. Table II shows the values of the weights for four action types defined in this work, as well as some examples in the applications of tidying up a workshop, cooking and washing clothes. A great number of tasks in a collaborative environment can be grouped into the defined action types, both in industrial and household settings. For simplicity, we only show only four of the possible action types, proving the concept and the potential of using an action type model. This can be easily extended to a larger number of types, as well as applied to different agent models. The main benefit of developing this action type model is that it can be used to develop an action outcome simulator, defining the probabilities of successful execution of an action type by a certain agent, as presented in the following section.

V. ACTION OUTCOME SIMULATOR

Our ultimate goal is to be able to learn appropriate action costs associated with an agent state during a collaboration, so that a successful plan with the right action assignment can be generated and executed. Collecting good sets of learning examples is crucial, but obtaining data in the real world is very difficult and time-consuming. We intend to use an action outcome simulator in order to learn an initial set of action costs, which can eventually be refined in the real application. Note that the simulator is not intended to replace or match the real world, but to provide a tool for early evaluation of plans and for learning initial action costs. The simulator developed in this work determines an action's outcome (success or failure) when assigned to a particular agent in the plan. The probability of success of each action is defined based on the agent model and the action type model presented in the previous sections. Let's define an agent type ag using the agent state model $AS = \{K, D, S, F, V\}$ described in Table I. Applying the action type model from Sec. IV, the success probability of an action type AT can be written as

$$\begin{aligned} p\_success_{AT} = w_{K,AT} * K_{ag} + w_{D,AT} * D_{ag} + \\ w_{S,AT} * S_{ag} + w_{F,AT} * F_{ag} + w_{V,AT} * V_{ag} \end{aligned} \tag{4}$$

where $w_{K,AT}$, for example, represents the impact the Knowledge (K) element has on the success of action type AT, as defined by the action type model. The same applies to the rest of the elements in the agent model. This can be generalised to any agent state model and action type model: for an agent type ag defined using an agent state model AS comprising m elements (Eq. 1), and an action type model AM comprising the action type AT (Eqs. 2 and 3), the success probability of this action type is defined as:

$$p\_success_{AT} = \sum_{k=1}^{m} w_{E_{k,AT}} * E_{k_{ag}} \tag{5}$$

TABLE II
ACTION TYPES

Action Type	Description	Agent Model Impact Weights $\{w_K, w_D, w_S, w_F, w_V\}$	Example Actions in Different Applications		
			Tidying up a workshop	Cooking	Washing clothes
1	Actions requiring focus and previous knowledge of the task, but no physical effort.	{H, M, L, H, M}	Taking inventory.	Selecting ingredients.	Classifying clothes by washing temperature.
2	Actions requiring physical effort, but no focus.	{M, M, H, L, H}	Carrying a heavy box.	Kneading the dough.	Carrying a heavy bag of clothes.
3	Actions requiring strong focus, but not much previous knowledge or strength.	{L, M, M, H, H}	Carrying cutting tools.	Pouring ingredients on a hot pan.	Ironing clothes.
4	Actions requiring strength and dexterity, but not too much focus or specific knowledge.	{L, H, H, M, M}	Vacuuming the floor (heavy vacuum).	Grinding chillies with a pestle and mortar.	Hanging out wet clothes.

where $w_{E_{k,AT}}$ represents, for action type AT, the weight that the element E_k of the agent model has on the outcome of this action type, as defined by the action type model.

Once the success probabilities of all action types have been determined, the simulator can be implemented as:

```
for action in plan:
    get corresponding action type AT
    if p_success_AT <= random[0,1]
    then: action failure
    else: action success
```

Listing 1. Action Success Simulator

The higher $p\_success$ is, the less likely the condition for action failure is to occur. This provides a neat, simple and generalisable action outcome simulator which, based on a proper definition of an agent model and an action type model following the structure defined in this work, can be extremely useful for applications such as action cost learning in HRC.

VI. ACTION COST LEARNING

When integrating the agent states in the task planning action costs, the value given to these costs has a major impact on the action allocation in the generated plan, and therefore on its success. Manually estimating the costs is a challenging task, and can lead to unexpected and inappropriate plans for the current collaboration scenario and participating agents. We propose a fast and generic method to learn an initial set of Planning Domain Definition Language (PDDL) [29], [30] action costs associated with an agent type, so that actions can be suitably allocated in human-robot collaborative plans. The method uses the simulator developed in Sec. V, which can be based on any action and agent models specified as described in Sec. IV and Sec. III respectively.

Learning Algorithm. The learning is done through an evolutionary optimisation algorithm that evaluates a fitness function to select the "best" set of action costs, and therefore preferred actions, for the agent type being evaluated. An advantage of using this learning method is its generality, as no specific structural knowledge about planners or domains is needed. In our case, the method learns the action cost values from the plans generated using the POPF planner [31]. The action costs are defined in the PDDL planning domain as action effects increasing the total plan cost. The evolutionary algorithm selected for this work is the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [32], due to its suitability for black-box optimisation, where the derivative of the objective function is not known. Furthermore, there is no need to discretise or set bounds in the search. The implementation has been made using the open-source DEAP evolutionary computation framework [33]. The steps implemented for learning action costs using the CMA-ES algorithm and the action outcome simulator are as follows:

1) Generate a population of individuals (candidate solutions): sets made of costs for each action type ($[costAction_{type1}, costAction_{type2}, ...]$).
2) Evaluate every individual in the population.
   2.1 Use the individual to update the action costs in the planning knowledge base.
   2.2 Generate the PDDL problem including the action cost values being evaluated.
   2.3 Generate the plan.
   2.4 Parse the generated plan and extract the elements included in the fitness function.
   2.5 Calculate the fitness value for the individual.
3) Select parents from the fittest individuals.
4) Reproduce offspring of the next generation (recombination and mutation).
5) Repeat until a termination criterion is met.

Fitness Function. A fitness function is used to assess the goodness of each set of action costs evaluated during learning (step 2.5). From the simulator, we know which actions are more likely to be successful for each agent type. However, other elements that don't necessarily affect the outcome of the actions might play a role in the preferred plan generation, such as the time taken by the actions, the priority of the actions to be executed, etc. The fitness function defined to be minimised by the learning algorithm contains these elements, taking values in the range [0,1], where 0 is the optimal case:

- *goalNotReached*: determined by the action outcome simulator, takes a value of 0 if all actions are successful.
- *planLength*: plan length in seconds, normalised between 0 and 1, where 0 represents the minimum plan length.
- *magControl*: factor introduced to avoid cost values exploding during the search. The lower the values in the individual, the lower it is.
- *priorKnowledge*: helps the search by introducing prior knowledge about the relative values between the action costs. The factor is set to 0 if the relationship (bigger

Fig. 2. Agent types defined in terms of the agent model.

than or smaller than) between the action cost values in the individual corresponds to the one in the simulator's action success probabilities e.g., if action A has a higher success probability than action B, we would expect action A to have a lower cost than action B.

The elements are weighted based on how much importance they are given in the resulting plan using

$$\begin{aligned} fitness = w_{goalNotReached} * goalNotReached \\ + w_{planLength} * planLength \\ + w_{magControl} * magControl \\ + w_{priorKnowledge} * priorKnowledge \end{aligned} \quad (6)$$

where the sum of the weights equals 1. Based on how the factors are weighted, the learnt costs should generate plans that prioritise the plan length or success differently. Sec. VII-B shows examples of learning action costs for a number of agent types using different weights in the fitness function.

VII. Evaluation

A. Simulator Evaluation - User Survey

An unsupervised survey[1] has been conducted to evaluate how "realistic" and coherent the action outcome simulator is. The link was advertised and 40 people participated (N=40, mean age=28) without active recruitment. People were asked, for a number of agent types and applications, what their preferred actions in the collaborative plan would be. We then associate these actions to an action type, and verify if the preferred actions correspond to the action types with the highest success probabilities obtained by the simulator. Inspired by the work in [12], we have defined a number of "personas", or agent types, in terms of the elements in the agent model as shown in Fig. 2. Table III shows, for each agent type, the resulting action success probabilities from applying Eq. 4, and therefore the preferred action types for each agent based on the simulator.

The participants were given a description of each agent and were asked which two out of the actions in the application they believed the agent would prefer to execute. This was asked for the three applications presented in Table II,

[1]http://www.iri.upc.edu/groups/perception/#HRC_LearningCosts

TABLE III
Success probabilities for each action type and agent

	Simulator's success probabilities per action type $(p_{AT_1}, p_{AT_2}, p_{AT_3}, p_{AT_4})$	Preferred action types (based on simulator)
Joey	(0.80, 0.45, 0.60, 0.45)	1, 3
Rachel	(0.45, 0.80, 0.60, 0.67)	2, 4
Ross	(0.59, 0.47, 0.74, 0.51)	3, 1
Monica	(0.32, 0.66, 0.51, 0.74)	4, 2

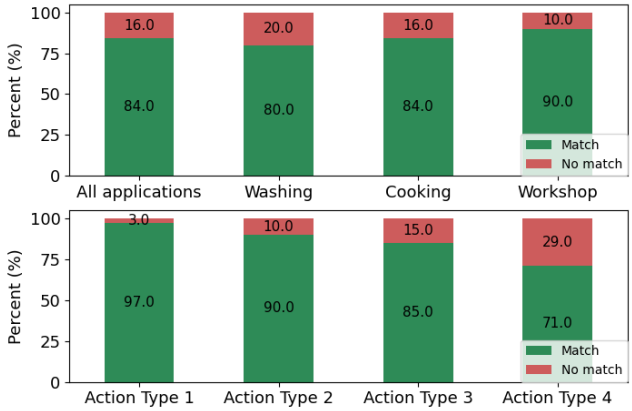

Fig. 3. Human Survey vs Simulator - comparing preferred actions by application (top) and action type (bottom)

namely cooking, washing clothes and tidying up a workshop. The evaluation consists in comparing the two action types chosen by the participants to the ones set as most likely to be successful by the simulator. Fig. 3 shows the percentage of times the simulator and human preferences would have matched, as well as the percentage of times the human preference would not have been reflected in the simulator.

The results are analysed by application (top) and by action type (bottom). It can be seen that in 84% of the cases, the simulator success probabilities match the choices a human would make based on their state. When considering the action types, even though the survey results match the simulator in most cases, the human preference for action type 4 is not caught by the simulator as well as for the other types. This suggests that the grouping of these actions into this action type might not be suitable, and a new action type would need to be defined. Nevertheless, the results show that the grouping of actions into action types works well in most cases, with the simulated success probabilities matching the human preferred actions in a number of different applications. Furthermore, this confirms that modelling the agent in terms of the elements defined in this work gives a sufficient representation of the human state for a choice on preferred actions to be made. To conclude, the action outcome simulator based on these models can be considered coherent, "realistic", and suitable as a starting point for action cost learning in AI task planning.

B. Action Cost Learning Evaluation

The intention of this evaluation is to reinforce the concept of learning PDDL action costs using a black-box optimisa-

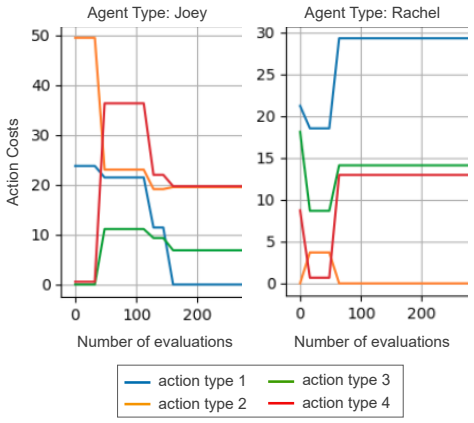

Fig. 4. Learning action costs for different agent types. Plan success is prioritised: action types with the highest success probabilities for the agent obtain the lowest costs and vice versa.

tion algorithm and a plan outcome simulator in HRC. Fig. 4 shows the results of learning action costs for two agent types, Joey and Rachel (defined in Fig. 2). The learning process takes an average of 190 minutes. For Joey, action type 1 has the highest success probability (0.80), followed by action type 3 (0.60), and action types 2 and 4 (0.45) (see Table III). When the plan success is prioritised, the learnt cost associated with type 1 should be the lowest, followed by type 3, with types 2 and 4 having the highest cost. This is supported by the results in Fig. 4. The same principle holds for the agent Rachel, showing that the method can learn coherent costs for different agent types. Note that the dynamic elements of the agent model could vary during a real execution, in which case the agent would become a new agent type, with different associated simulator success probabilities. This would trigger a replan, applying the action costs learnt using these different probabilities.

Varying the fitness function. We evaluate the effect of varying the weight values in the learning fitness function presented in Eq. 6. Let's take the cooking scenario as an example, where the four tasks described in Table II need to be distributed between the human and the robot. In this case, the human doesn't know the ingredients to be selected, and this action will therefore always fail in the simulator. The human takes 50 seconds longer than the robot to perform the rest of the tasks (kneading, pouring and grinding). If the plan success is strongly prioritised during learning, the resulting plan should not include the "select ingredients" action assigned to the human, as this would result in a failed, (even if shorter) plan. Instead, the learnt costs are expected to generate a longer but successful plan (Fig. 5a). Contrarily, if the plan length factor is strongly weighted in the fitness function, the learnt costs should result in a shorter, even if unsuccessful plan (Fig. 5b). The four sets of fitness function

TABLE IV

WEIGHTS IN FITNESS FUNCTION FOR EACH CASE EVALUATED

	$w_{goalNotReached}$	$w_{planLength}$	$w_{magControl}$	$w_{priorKnowledge}$
Case 1	0.85	0.00	0.05	0.10
Case 2	0.65	0.20	0.05	0.10
Case 3	0.40	0.45	0.05	0.10
Case 4	0.25	0.60	0.05	0.10

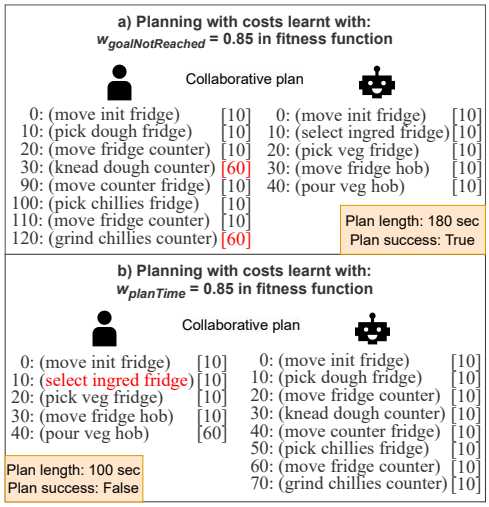

Fig. 5. Collaborative plans resulting from action costs learnt with different weights in the learning fitness function. The human doesn't know the recipe, causing the select function to fail (0% success probability in simulator). The human takes longer to execute the other tasks. Success is prioritised in case a), whilst plan length is prioritised in case b).

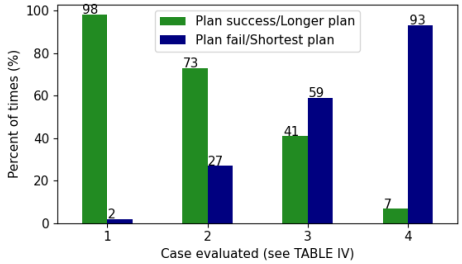

Fig. 6. Comparing the times the plan generated from planning with the learnt costs correspond to a successful but longer or shortest but failed plan, for different weights in the learning fitness function (see cases in Table IV).

weights from Table IV are evaluated. We run each case 40 times, and compare the percentage of times the plan resulting from planning with the learnt action costs corresponds to a successful but longer plan, or the shortest but failed plan.

Fig. 6 shows how if no importance is given to the plan length (Case 1), the learnt costs generate a plan that doesn't include the action with 0% success probability, even if it takes longer to execute. As $w_{planLength}$ is increased and $w_{goalNotReached}$ is decreased from cases 2-4, the time factor is more frequently prioritised, trading the success probability of the plan for a shorter plan.

These results highlight the possibility of learning coherent PDDL action costs for a set of agent types using a generic learning framework and action simulator. Depending on the application priorities, different weights can be set and new factors can be added to the fitness function to learn costs that reflect these preferences in the plan generated.

VIII. CONCLUSIONS AND FUTURE WORK

In this paper, we develop a framework to learn an initial set of action costs for task planning in HRC scenarios, influencing the action allocation based on the agent states. The learning is done using a simulator, facilitating and accelerating posterior fine-tuning in real applications. The

framework integrates a number of novel elements, including an agent state model inspired by RPG, the concept of an action type model, and an action outcome simulator setting the action success probabilities for an agent type based on the models. The simulator is used as a black-box function in an evolutionary algorithm to learn a set of action costs associated with a number of agent types and states, considering their "preferred" actions. The validity of the simulator is supported by a user survey, and the learning results show that coherent action costs can be learnt for different agent types and requirements in the plan. The resulting system is a generic framework integrating components easily extendable to different applications, models, and planning formalisms.

The framework will require evaluation in real scenarios, which will allow for the refinement of the agent and action models. This will in turn improve the simulator's effectiveness. A larger number of action types will be defined, and we suggest querying a language model such as GPT-4 [34] to classify new actions into the defined action types, by providing a description of these. In terms of the learning component, different parameters and learning algorithms could be evaluated. Nevertheless, this constitutes a working proof of concept for learning action costs using a simulator in HRC applications, facilitating the adaptability of the plan to the contributing partners in the initial deployment stage.

REFERENCES

[1] S. Izquierdo-Badiola, G. Canal, C. Rizzo, and G. Alenyà;, "Improved task planning through failure anticipation in human-robot collaboration," in *International Conference on Robotics and Automation (ICRA)*, 2022, pp. 7875–7880.

[2] P. Khandelwal, F. Yang, M. Leonetti, V. Lifschitz, and P. Stone, "Planning in action language bc while learning action costs for mobile robots," *International Conference on Automated Planning and Scheduling*, 2014.

[3] S. Jiménez, F. Fernández, and D. Borrajo, "Integrating planning, execution, and learning to improve plan execution," *Computational Intelligence*, vol. 29, 02 2013.

[4] M. Fiore, A. Clodic, and R. Alami, "On Planning and Task achievement Modalities for Human-Robot Collaboration," in *International Symposium on Experimental Robotics (ISER 2014)*, Marrakech, Morocco, Jun. 2014, p. 15p.

[5] S. Devin, A. Clodic, and R. Alami, "About decisions during human-robot shared plan achievement: Who should act and how?" in *Social Robotics*. Springer International Publishing, 2017, pp. 453–463.

[6] S. Devin and R. Alami, "An implemented theory of mind to improve human-robot shared plans execution," *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pp. 319–326, 2016.

[7] G. Milliez, R. Lallement, M. Fiore, and R. Alami, "Using human knowledge awareness to adapt collaborative plan generation, explanation and monitoring," in *11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2016, pp. 43–50.

[8] O. Görür, B. Rosman, G. Hoffman, and S. Albayrak, "Toward integrating theory of mind into adaptive decision- making of social robots to understand human intention," in *HRI Workshop on the Role of Intentions*, 2017.

[9] O. C. Görür, B. Rosman, F. Sivrikaya, and S. Albayrak, "Social cobots: Anticipatory decision-making for collaborative robots incorporating unexpected human behaviors," in *ACM/IEEE International Conference on Human-Robot Interaction*, 2018, p. 398–406.

[10] O. C. Görür, B. Rosman, and S. Albayrak, "Anticipatory bayesian policy selection for online adaptation of collaborative robots to unknown human types," in *International Conference on Autonomous Agents and MultiAgent Systems*, 2019, p. 77–85.

[11] A. Cooper and P. Saffo, *The Inmates Are Running the Asylum*. USA: Macmillan Publishing Co., Inc., 1999.

[12] A. Andriella, C. Torras, and G. Alenyà, "Learning robot policies using a high-level abstraction persona-behaviour simulator," in *28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, 2019, pp. 1–8.

[13] J. C. Mike Mearls, *Dungeons & Dragons Player's Handbook*. Wizards of the Coast, 2014.

[14] J. Chambers, *Cortex System Role Playing Game*. Diamond Comic Distributors, 2009.

[15] C. V. Cam Banks, Rob Donoghue, *Leverage: The Roleplaying Game*. Margaret Weis Productions, 2010.

[16] S. Jiménez, T. De la Rosa, S. Fernandez, F. Fernández, and D. Borrajo, "A review of machine learning for automated planning," *The Knowledge Engineering Review*, vol. 27, 12 2012.

[17] M. Castilho, L. Künzle, E. Lecheta, V. Palodeto, and F. Silva, "An investigation on genetic algorithms for generic strips planning," vol. 3315, 11 2004, pp. 185–194.

[18] J. Levine and D. Humphreys, "Learning action strategies for planning domains using genetic programming," in *Applications of Evolutionary Computing*. Springer Berlin Heidelberg, 2003, pp. 684–695.

[19] R. Aler, D. Borrajo, and P. Isasi, "Using genetic programming to learn and improve control knowledge," *Artificial Intelligence*, vol. 141, no. 1, pp. 29–56, 2002.

[20] E. Quintero, V. Alcázar, D. Borrajo, J. Fernández-Olivares, F. Fernández, A. Garcia-Olaya, C. Guzman, E. Onaindía, and D. Prior, "Autonomous mobile robot control and learning with the pelea architecture," in *Automated Action Planning for Autonomous Mobile Robots*, 2011.

[21] G. Canal, G. Alenyà, and C. Torras, "Adapting robot task planning to user preferences: an assistive shoe dressing example," *Autonomous Robots*, vol. 43, no. 6, pp. 1343–1356, Aug 2019.

[22] B. Silverman, "Toward realism in human performance simulation," *Advances in Human Performance and Cognitive Engineering Research*, vol. 5, 12 2004.

[23] M. Carroll, R. Shah, M. K. Ho, T. L. Griffiths, S. A. Seshia, P. Abbeel, and A. D. Dragan, "On the utility of learning about humans for human-ai coordination," *CoRR*, vol. abs/1910.05789, 2019.

[24] D. Strouse, K. R. McKee, M. M. Botvinick, E. Hughes, and R. Everett, "Collaborating with humans without human data," *CoRR*, vol. abs/2110.08176, 2021.

[25] A. Favier, P.-T. Singamaneni, and R. Alami, "An Intelligent Human Simulation (InHuS) for developing and experimenting human-aware and interactive robot abilities," Jun. 2021.

[26] A. Favier, P. T. Singamaneni, and R. Alami, "An intelligent human avatar to debug and challenge human-aware robot navigation systems," in *ACM/IEEE International Conference on Human-Robot Interaction*, ser. HRI '22. IEEE Press, 2022, p. 760–764.

[27] R. Kosti, J. M. Alvarez, A. Recasens, and A. Lapedriza, "Context based emotion recognition using emotic dataset," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 11, pp. 2755–2766, 2020.

[28] C. Zaga, K. Truong, M. Lohse, and V. Evers, "Exploring child-robot engagement in a collaborative task," in *Child-Robot Interaction Workshop: Social Bonding, Learning and Ethics*. Lisbon, Portugal: INESC-ID, 2014, p. 3.

[29] M. Ghallab, A. Howe, C. Knoblock, D. Mcdermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL—The Planning Domain Definition Language," 1998.

[30] M. Fox and D. Long, "PDDL2.1: An extension to PDDL for expressing temporal planning domains," *Journal of artificial intelligence research*, vol. 20, pp. 61–124, 2003.

[31] A. Coles, A. Coles, M. Fox, and D. Long, "Forward-chaining partial-order planning," in *International Conference on Automated Planning and Scheduling*, 2010, pp. 42–49.

[32] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[33] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, jul 2012.

[34] OpenAI, "Gpt-4 technical report," 2023.