2024 IEEE 20th International Conference on Automation
Science and Engineering (CASE)
August 28 - September 1, 2024. Bari, Italy

Preference Elicitation and Incorporation for Human-Robot Task Scheduling

Neel Dhanaraj[1], Minseok Jeon[1], Jeon Ho Kang[1], Stefanos Nikolaidis[1], and Satyandra K. Gupta[1]

Abstract— **In this work, we address the challenge of incorporating human preferences into the task-scheduling process for human-robot teams. Humans have various individual preferences that can be influenced by context and situational information. Incorporating these preferences can lead to improved team performance. Our main contribution is a framework that helps elicit and incorporate preferences during task scheduling. We achieve this by proposing 1) a constraint programming method to generate a range of plans, 2) an intelligent approach for selecting and presenting task schedules based on task features, and 3) a preference incorporation method that uses large language models to convert preferences into soft constraints. Our results demonstrate that we can efficiently generate diverse plans for preference elicitation and incorporate them into the task-scheduling process. We evaluate our framework using an assembly-inspired case study and show how it can effectively incorporate complex and realistic preferences. Our implementation can be found at github.com/RROS-Lab/Human-Robot-Preference-Planning.**

I. Introduction

Humans exhibit a wide range of personalized preferences when collaborating with robot teammates on tasks. These preferences are often influenced by the cognitive and physical demands associated with the tasks. For instance, a task that is cognitively (or physically) challenging for one individual may be relatively easy for another. Preferences therefore are highly individualized. Some may opt to tackle challenging tasks first and then proceed to easier ones, while others might prefer the reverse. Additionally, humans may possess contextual preferences based on situational awareness. For example, they may prefer to defer certain tasks in anticipation of delayed material or tool arrivals. Similarly, they may anticipate periods of unavailability and prefer to avoid specific tasks during those times.

There have been many efforts for learning or eliciting human preferences, such as learning a human preference reward function from planning demonstrations or showing humans potential plans, and actively soliciting a score/rank of the plans. These methods are helpful when humans cannot express their preferences but can require a significant amount of feedback. However, preferences can be highly situational and individualized and may not be able to be specified beforehand. Instead, we explore a framework in which we iteratively generate plans, elicit individualized preferences from humans, and intelligently incorporate them.

To elicit human preferences, we first need to inform humans of the possible planning space. We can naturally pose this as a diverse planning problem where we show humans a set of possible plans that are both high in quality and diverse. We propose a constraint programming formulation for diverse planning designed to produce a set of high-quality, diverse solutions. Humans naturally compare different schedules based on specific features of interest. However, showing an exhaustive list of all possible solutions can lead to cognitive overload. To mitigate this, we aim to identify and present a subset of diverse solutions within the task feature space. This is accomplished by clustering the solution set as a function of task features of interest.

[1]University of Southern California, Los Angeles, CA, USA.

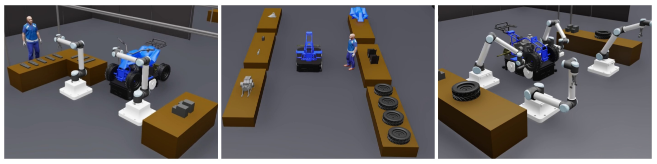


Fig. 1: An example of a human multi-robot assembly cell for producing all-terrain vehicles in high-mix low-volume has been developed in simulation. When coordinating all agents, task scheduling methods must take into account human preferences, as well as dynamic changes in task requirements and the environment.

The second problem is incorporating the preferences into the constraint programming formulation. We propose to view preferences as soft constraints which we must try to satisfy. Using constraint programming enables us to propose very expressive preferences using logic formalism; this allows us to formulate complex preferences as soft constraints. Incorporating the preferences during planning becomes a problem of finding solutions that minimize the initial objectives and the number of violated soft constraints. Converting human preferences into mathematical soft constraints is now practical with recent success in large language models.

Our key idea in this work is to couple these two perspectives as an iterative planning approach. We iteratively generate diverse plans to present to the human in order to elicit preferences.

1) We present a constraint programming formulation to enable diverse plan generation for multi-robot task scheduling in order to present planning options to the human
2) We show a method for formulating human-specified preferences as soft constraints and show that large language models can convert preferences into soft constraints
3) We present results that show the effectiveness of the proposed iterative planning in a real-world case study.

II. Related Works

Planning with Preferences: There has been considerable work in planning for complex domains while considering preferences. This is useful, especially in human-robot collaboration. Experimentation has shown that human agents would rather work with robots that account for their preferences

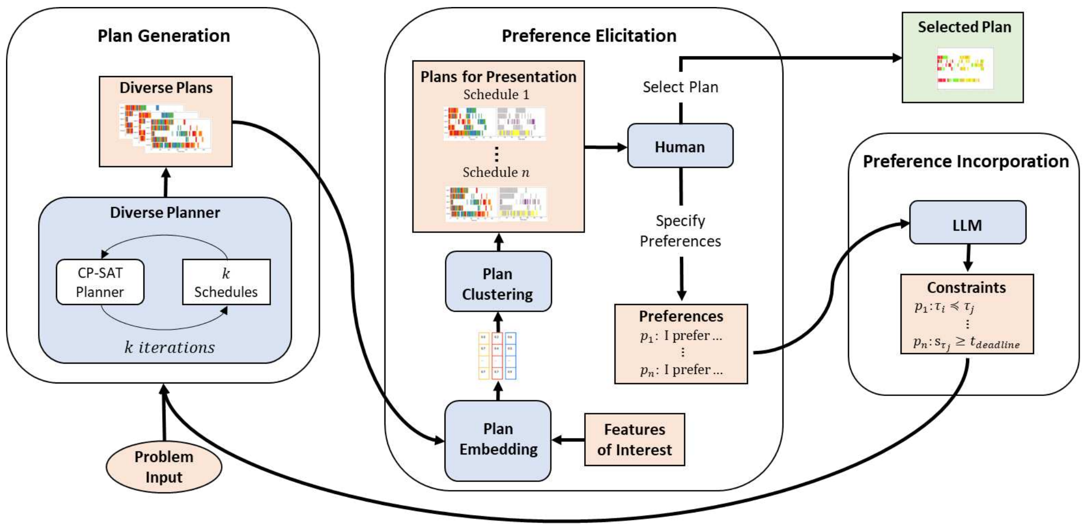


Fig. 2: This is a system diagram illustrating the process of generating diverse plans, selecting and presenting them to the human for preference elicitation, and incorporating preferences as soft constraints using large language models

[1], [2]. However, this can be challenging due to the large number of preferences that may exist [3]. Preferences may be very context-dependent and difficult to express as a static model input [4], [5]. For instance, when planning for the Mars Exploration Rover, multiple science and engineering groups meet to generate the plan for the rover [6]. Meeting all subteam preferences, as well as rover system constraints, can be very challenging, and automated planning has the potential to alleviate these challenges. The broad area of preference-based planning has been extensively studied as an extension of the classical planning problem. One well-known method to plan with preferences is to formulate them as soft constraints, where a preference incurs a cost if it is not met [7]. Traditional planning techniques such as MDP solving methods [8], [9], constraint satisfaction [10], [11], and HTN planning methods [12] can then be adapted to find the minimum cost solution. Many methods assume that a preference model is specified. However, in many cases, preferences need to be elicited from humans during the planning process. Iterative planning or mixed-initiative planning is an extension that involves allowing users to naturally specify and utilize constraints during planning. This approach includes generating multiple qualitatively different plans and providing detailed explanations of those plans for iteration. [7], [13]–[15]

Eliciting and Learning Preferences Eliciting preferences as constraints are an integral part of iterative planning [16]. Iterative planning shares common goals with explainable planning [17], [18], as explainable AI is used to explain different aspects of the planning process so that humans can understand the effect of enforcing different preferences. Explainable AI in planning aims to provide frameworks to explain why a planner came to or did not come to a specific solution, why certain solutions are not feasible, as well as why certain solutions are better than what the human would intuitively do [17]. This is used to iteratively elicit preferences from the human user. Defining user preferences as constraints can often be challenging [19]. Therefore, new works have investigated ways to learn preferences implicitly from demonstrations. One approach involves clustering multiple-user demonstrations using inverse reinforcement learning to learn user preferences offline [1], [20], [21]. A new user can then provide a demonstration that can be matched to a dominant preference group, and the preference model can be used in the planning process.

III. Problem Formulation

We begin with the problem formulation for multi-robot task scheduling with task ordering constraints. We base our formulation on our prior work for contingency-aware task allocation and scheduling [22]. We begin with a set of n agents $\mathcal{A}_i \in N$ for which there is one human in the team of agents. Furthermore, there is a set of m tasks $\tau_j \in T$ that each has a processing time p_j. Each task is designated for a specific type of agent, and each agent has a subset of tasks that they are responsible for. We take inspiration from complex assembly tasks, where task ordering is critical. To represent these constraints, we use a hierarchical task network (HTN), which is a tree with a root node that groups all the tasks, and leaf nodes that represent individual tasks. Non-leaf nodes are subtask groups that can be further broken down and specify ordering constraints. For instance, some tasks must be executed sequentially, while others can be executed concurrently. We use an HTN because it is an expressive way to represent assembly task constraints. However, during the planning process, we decompose the HTN into P, a set of task precedence pairs (i, j) specifying that τ_i must precede τ_j.

Our traditional objective is to find the best plan π from all the possible plans Π that have low-cost $C(\pi)$. Specifically, for this application, we want to create a task schedule that assigns start times for all tasks and minimizes the time it takes to execute all tasks. However, we also want to consider the preferences of the human partners. Our assumption is that there are multiple candidate plans that fall within an acceptable cost range, and the human's preferred plan is one that is within this cost range but maximizes the satisfaction of their preferences. To determine the human's preferred plan, we need to present them with k diverse plans, which they can use to specify their preferences and choose their preferred plan. Our objective is to find a subset of low-in-cost (high-quality) and high-in-diversity metric $D(\Pi_d)$ (diverse) plans $\Pi_k \subseteq \Pi$. This is illustrated in the following objective.

$$\underset{\Pi_k \subseteq \Pi}{\operatorname{argmin}} \sum_{\pi \in \Pi_k} Ac(\pi) - Bd(\Pi_k) \tag{1}$$

We model preferences as constraints the plan must satisfy. The human can then specify preferences as hard constraints, such as tasks must be completed by a deadline, or soft constraints, where there is an incurred penalty for an unmet preference. The focus of this work is handling preferences as soft constraints. A preference for a subset of tasks will result in a set of soft constraints. We indicate the satisfaction of each constraint with either a) booleans whose value is 1 if the constraint hasn't been met else 0, and b) slack variables, which quantify to what degree the constraint hasn't been met. We then map how much the preference (set of constraints) for a subset of tasks is being violated to a penalty term λ_i; furthermore, some preferences may be more important than others; we define a weight w_i, which indicates the importance of the preference. Overall, our objective becomes determining Π_k, which is of 1) high quality such that it minimizes makespan and the degree of unmet preference soft constraints and 2) of high diversity. Our objective then becomes the following:

$$\underset{\Pi_k \subseteq \Pi}{\operatorname{argmin}} \sum_{\pi \in \Pi_k} [Ac(\pi) + \sum_{\lambda_j \in \Lambda} w_j \lambda_j(\pi_k)] - Bd(\Pi_k) \tag{2}$$

A. Overview of Approach

In this work, we introduce a framework for integrating human preferences into task scheduling for human-robot collaboration as shown in Figure 2. Our approach is iterative, consisting of three main stages: diverse plan generation and preference elicitation and incorporation. Initially, we generate a diverse set of high-quality task schedules using a constraint programming formulation (Section IV). To efficiently elicit preferences, we present the human operator with a representative subset of these schedules. This subset is selected based on task features of interest and the diversity of the schedules, which is determined through graph embeddings and hierarchical clustering (Section V). The operator provides feedback on these schedules, expressing preferences that are then translated into soft constraints using natural language processing techniques (Section VI). This process iterates, with each step further refining the schedules based on updated preferences until a satisfactory schedule is identified.

IV. Generating Diverse Plans Via Constraint Programming

We need a diverse planning method that can handle the combinatorial complexity of task scheduling to ensure that the solutions generated are of high quality while efficiently producing a diverse set of solutions. One can measure a solution set diversity as the distance between all of the solutions. To find a set of k diverse and high-quality solutions, an exact method would be to solve a planning problem with k copies of decision variables and calculate values for all variables that maximize both quality and how diverse the plans are in comparison to each other. However, this leads to an increase in computational complexity as the number of decision variables and constraints increases with k. Instead, we can use an iterative, greedy approach proposed by prior research [23] to approximately solve the diverse planning problem. This involves calling a classical planner for k iterations and, in each iteration, finding one best solution that maximizes both the quality and distance measure between solutions found in previous iterations.

A. Constraint Programming Formulation

We explore solving a mathematical program using a Constraint Programming-Satisfaction (CP-Sat) approach as our classical planner of choice. CP-Sat solvers find good solutions quickly and allow us to easily incorporate complex human preferences as soft constraints using predicate logic formalism. The input for our program is the task set, an HTN, a set of plans Π_p that we found from previous iterations, and an additional set of hard and soft constraints $\{C, \Lambda\}$ which the human can specify. More details on formulating preferences as soft constraint penalties are provided in Section VI. Our mathematical constraint program is proposed below.

Input$(T, \text{HTN}, \Pi_p, \Lambda, C)$:
$P \leftarrow \text{HTN}$

$$\text{Minimize} \quad At + \sum_{\lambda_l \in \Lambda} w_l \lambda_l - B \sum_{\pi_k \in \Pi_p} D(\pi, \pi_k) \tag{3}$$

subject to

$$s_j \geq e_i, \qquad \forall (i,j) \in P \tag{4}$$

$$\text{NoOverlap}(\{o_i | i \in T_k\}), \qquad \forall k \in N \tag{5}$$

$$t = \text{Max}(\{e_i | i \in T\}) \tag{6}$$

$$c \in C \tag{7}$$

$$D(\pi, \pi_k) = \sum_{s_i \in \pi, s_{ik} \in \pi_k} |s_i - s_{ik}| \tag{8}$$

We first extract a set of precedence constraints P for task pairs from the HTN. Equation 4 constrains task j to start after task i for all of the precedence pairs. $o_i(s_i, p_i, e_i)$ is an interval variable for task i, which enforces $s_i + p_i = e_i$ and is also sequencing variable used in the $NoOverlap(.)$ scheduling constraint. Equation 5 specifies that each agent's set of task interval variables o_i cannot overlap. Equation 6 defines the makespan as the maximum of the task end times. Equation 7 includes other human-given constraints that can further constrain the planning space. Lastly, we quantify the distance between the two plans as the Manhattan distance of all task start times in both plans.

The objective of the task scheduling problem is to minimize the weighted sum of three terms: the makespan, soft constraint violation penalties, and the negative sum of distance measures between the current plan and all plans found in previous iterations. This is defined by Equation 1.

B. Generating Diverse Plans

We now describe our iterative diverse planner using Algorithm 1. We begin the algorithm by first computing an initial best plan π_b by minimizing just the schedule makespan (Line 4). This allows us to initialize Π_b (the set of best diverse plans found in previous iterations) and Π_{All} (the set of all encountered plans during solving process) with π_b and Π_{iter} respectively (Line 5,6). We also add a constraint that bounds the makespan t for the next planning iterations to be almost suboptimal by a factor of α compared to the best makespan found in the zeroth iteration (Line 7). The value of α is provided by the user, and it ensures that the encountered plans of the current iteration Π_{iter} are of high quality. Finally, we run the planner for k iterations (Line 8). In each iteration, we find a plan that has a low makespan and is also at a high distance from the plans found in the previous iterations (Line 9). We then update Π_b with the best plan found in that iteration and add all the plans found to Π_{all} (Line 10,11). When showing the user the available options, we record all plans found so that there is a good population of solutions to select from.

Algorithm 1

```
 1: $\Pi_b$: Set of best diverse plans found in previous iterations
 2: $\Pi_{all}$: All encountered plans
 3: function DIVERSEPLANGENERATION($T$, HTN, $\Lambda$, $C$)
 4:     $\alpha$: Sub-optimality Factor
 5:     $\pi_b, \Pi_{iter} \leftarrow$ PLANNER($T$, HTN, $\Lambda$, $C$)
 6:     $\Pi_b \leftarrow \{\pi_b\}$
 7:     $\Pi_{all} \leftarrow \Pi_{iter}$
 8:     $C \leftarrow C \cup \{t \leq \alpha * t_{\pi_b}\}$
 9:     while $|\Pi_p| \leq k$ do
10:         $\pi_b, \Pi_{iter} \leftarrow$ PLANNER($T$, HTN, $\Pi_b$, $\Lambda$, $C$)
11:         $\Pi_b \leftarrow \Pi_b \cup \{\pi_b\}$
12:         $\Pi_{all} \leftarrow \Pi_{all} \cup \Pi_{iter}$
        return $\Pi_b, \Pi_{all}$
```

V. Plan Selection and Presentation for Elicitation

It is crucial to present a diverse and representative subset of potential task schedules to elicit preferences from a human operator. Given a large solution space, directly presenting all feasible schedules to the human operator can lead to cognitive overload. To address this challenge, we propose a methodology that leverages graph embeddings and hierarchical clustering to identify distinct schedules. By focusing on task features of interest to the human operator, this approach enables the selection of a diverse subset of schedules that are likely to elicit meaningful preferences, thereby facilitating more effective human-robot collaboration.

To capture the structural characteristics of each schedule, we represent schedules as directed acyclic graphs where nodes correspond to tasks and edges represent task orderings. We model each node to have a vector of task features. We then employ the graph2vec algorithm [24] to generate embeddings for these schedule graphs. Graph2vec is particularly well-suited for this task as it is capable of capturing substructural information, which is essential for representing the human operator's preferences related to specific task features or sequences. The resulting embeddings serve as a compact representation of each schedule, preserving the essential structural information while enabling efficient comparison and clustering.

With the graph embeddings of the schedules obtained, we apply hierarchical clustering to group similar schedules together. This clustering step aims to identify distinct schedule clusters that represent the diverse solution space. By selecting a representative schedule from each cluster, we can ensure that the presented subset of schedules captures a wide range of planning options, thereby facilitating the elicitation of diverse and informative preferences from the human operator. The hierarchical nature of the clustering algorithm allows for flexibility in determining the granularity of the clusters, enabling the selection of a subset of schedules that balances diversity with the cognitive load on the human operator.

VI. Preference Incorporation

After receiving preferences expressed in natural language, we incorporate them into the planning problem by converting them into soft constraints for the mathematical program. We propose a prescriptive approach for converting preferences into associated decision variables and soft constraints, with illustrative examples provided. Additionally, we show how to integrate the approach with a large language model to automatically generate soft constraints.

A. Converting Preferences to Soft Constraints

To incorporate preferences into our scheduling application, we follow a four-step approach. Firstly, we extract the relevant tasks based on preference. We assume that all preferences for our scheduling application are based on specifying soft constraints over a collection of disjoint sets of tasks, where each set contains tasks that share a common feature. For instance, for preference 1 *"Tasks with feature A should precede tasks with feature B"*, we would extract two task sets, T_a and T_b.

Once we have collected the relevant tasks, we create new indicator variables that indicate the satisfaction of each constraint. There are two types of indicators: 1) boolean variables, which are true if the constraint is satisfied, and 2) integer slack variables, which quantify the degree of satisfaction. For instance, for the given preference, we use boolean variables x_{ij} to indicate whether task $i \in T_a$ precedes task $j \in T_b$.

Next, we formulate the soft constraints. For the above preference, the soft constraint would be $s_i + p_i \leq s_j \Rightarrow x_{ij} = 1$, where s_i and p_i represent the start time and processing time of task i, respectively. These constraints are often preference-specific and generally require domain expertise for formulation. However, we demonstrate that large language models can be used to automate the generation of these constraints from natural language preferences.

Finally, we define a penalty term λ associated with the indicator variables. This penalty term is incorporated into the objective function and is typically a sum, max, or min of the indicator variables, depending on the nature of the preference. For the first preference, the penalty function is a sum of the boolean variables $\lambda_1 = \sum_{i \in T_a, j \in T_b} w_{ij} \cdot (1 - x_{ij})$, where w_{ij} represents the importance of the preference. We show this methodology to two different preferences, including the aforementioned preference, to demonstrate its versatility in converting a wide range of preferences into soft constraints.

Preference 1: Tasks with feature A should precede tasks with feature B

Variables of Interest:

- Let $T_a \subseteq T$ be the subset of task indices that have feature A (tasks of type a).
- Let $T_b \subseteq T$ be the subset of task indices that have feature B (tasks of type b).

New Variables:

- For each task $i \in T_a$ and each task $j \in T_b$, introduce a binary variable x_{ij} which is 1 if task i precedes task j, and 0 otherwise.
- Weight term w_{ij}

Soft Constraints:

- For each task $i \in T_a$ and each task $j \in T_b$, create optional precedence constraints using reified constraints:

$$s_i + p_i \leq s_j \Rightarrow x_{ij} = 1, \quad s_j + p_j \leq s_i \Rightarrow x_{ij} = 0$$
$$\forall i \in T_a, \, \forall j \in T_b$$

this constraint specifies that x_{ij} is 1 if and only if tasks i precedes tasks j.

Penalty Term:

- The penalty term is defined as:

$$\lambda_1 = \sum_{i \in T_a, j \in T_b} w_{ij} \cdot (1 - x_{ij})$$

Preference 2: Tasks with feature D should be completed as early as possible

Variables of Interest:

- $T_d \subseteq T$: The subset of task indices that have feature D.

New Variables:

- None is required; the end time variable implicitly can be used as an integer indicator.

Soft Constraints:

- No constraints required

Penalty Term:

- Define the penalty term λ_2 as the maximum end time among tasks with feature D:
$$\lambda_2 = \max_{i \in T_d} \{e_i\}$$
- This term directly maps λ to the maximum end time of tasks with feature D, encouraging them to be completed as early as possible.

B. Large-Language Models (LLMs) for Incorporating Preference Soft Constraints

We use LLM to convert human preference in natural language to soft constraints in the code format. We use the method outlined in [25], [26]. We use GPT-4 as our backbone architecture to enable the model to consider enough example constraints and instructions in the prompt to incorporate humans' natural language input.

In the prompt, we first inform LLM that it will perform the assistive task of converting human natural language to Pythonic code. We also provide LLM with predefined function primitives to enable it to generate code that fits our coding practice, similar to [25], [26]. We then give the model initial example pairs of human preferences and corresponding codes for reference. Defined primitives include:

- **initialize**: Takes task feature as variable and outputs task sets that belong to the feature and indicator variables created. It initializes the formulation of constraints by fetching all the tasks belonging to the feature of interest. Then, it creates the indicator variable.
- **create_slack_variables**: Takes variable name, lower bound, and upper bound as input and creates and returns a new integer variable to formulate slack variable.
- **get_task_variables**: Takes task_id as input and fetches task variables such as start, end, and duration time.
- **apply_constraint**: Adds the soft constraint back to the objective function.

Finally, during execution, we provide LLM with the newly seen preference from humans, and LLM successfully generates the soft constraint incorporated into the objective function as a penalty term. An example of such a constraint generation is shown in Fig. 3. In this example, humans prompt LLM that tasks of type C should be done as close to each other as possible.

VII. Results

A. Problem Setup

We formulate an assembly task problem that requires a team consisting of four robots and one human to complete. The problem is based on the assembly of ATVs and satellites, which are typically high-mix and low-volume products. In our previous work, we have examined this type of assembly in detail [22], [27], [28]. Multiple subassemblies need to be assembled to complete an assembly task. Each subassembly requires a sequence of atomic tasks to be completed. These subassemblies can be worked on simultaneously and then must be assembled sequentially.

To represent this structure, we have created an HTN and assigned various preference-related feature attributes to the tasks. We randomly generated the atomic task duration, with a range of 10-25 time units for each assembly task. We have grouped four assembly tasks together, each containing the same subassemblies and atomic tasks that can be performed in parallel. Our main goal is to generate a final task schedule that completes multiple assemblies simultaneously while also taking into consideration various preferences related to the

					Quality			Diversity		
# Tasks	# Constraints	# Solutions			Makespan			Normalized L1		
Iteration:		1	5	10	1	5	10	1	5	10
50	50 (low)	121	485	907	(198.42, 6.82)	(197.83, 5.03)	(197.96, 5.37)	(5.86, 3.87)	(11.55, 7.40)	(12.91, 8.16)
	114 (med)	23	88	163	(266.04, 10.44)	(274.375, 18.28)	(277.35, 19.17)	(3.29, 4.41)	(8.68, 5.20)	(9.23, 4.96)
	314 (high)	17	87	155	(416.47, 18.36)	(415.87, 19.95)	(415.31, 18.43)	(3.66, 2.45)	(3.50, 2.59)	(3.39, 2.54)
100	100 (low)	138	766	1475	(384.08, 1.02)	(386.44, 7.79)	(385.62, 7.28)	(8.27, 6.14)	(14.26, 6.71)	(14.82, 6.33)
	228 (med)	132	746	1587	(421.66, 24.79)	(423.80, 26.30)	(425.77, 24.17)	(8.25, 5.04)	(11.19, 3.85)	(11.84, 3.86)
	628 (high)	105	503	1103	(603.6, 20.39)	(606.49, 25.47)	(606.7, 24.52)	(3.29, 2.67)	(8.51, 5.04)	(8.57, 4.80)
150	150 (low)	67	351	782	(577.79, 13.94)	(578.30, 12.41)	(578.93, 12.40)	(4.30, 3.04)	(12.31, 5.27)	(12.83, 4.56)
	342 (med)	122	461	887	(632.67, 17.93)	(626.47, 42.32)	(625.76, 47.75)	(7.33, 4.09)	(12.73, 4.93)	(13.75, 4.39)
	942 (high)	101	500	1014	(833.86, 28.81)	(825.45, 28.55)	(821.29, 33.14)	(4.98, 2.88)	(10.07, 4.52)	(10.82, 4.88)

TABLE I: Evaluation of diverse plan generation method on different problem inputs of varying task size and constraint level

```
def create_constraint(task_sets, indicator_variables):
    feature = ['C']
    # initialize
    task_sets, indicator_variables = initialize(feature)
    for i in task_sets['C']:
        for j in task_sets['C']:
            if i != j:
                # Define new variables
                var_name_diff = f"diff_{i}_{j}"
                var_name_abs = f"abs_{i}_{j}"
                diff_ij = create_slack_variable(var_name_diff,
                -100000, 100000)
                diff_ij_abs = create_slack_variable(var_name_abs,
                0, 100000)
                # Create equations
                task_i_end = get_task_variables(i)[2]
                task_j_start = get_task_variables(j)[0]
                # calculate the difference between the tasks
                model.Add(diff_ij == task_j_start - task_i_end)
                # take the absolute value of the difference
                model.AddAbsEquality(diff_ij_abs, diff_ij)
                # Link the absolute difference variable
                # to an indicator variable
                indicator_variables[(i, j)] = diff_ij_abs
    # Apply soft constraint
    apply_constraint()
```

Fig. 3: Example of LLM generated constraint program. In the code, LLM successfully initializes the constraint generation and sets the feature of interest 'C'. Then, it successfully recognizes the need for slack variables and generates them using the pre-defined function. Finally, it adds absolute equality constraints between tasks to incorporate it back into the objective function in 'apply_constraint().'

task. We will provide more details about these preferences in the case study.

Our diverse plan generation generation algorithm was implemented using OR-Tools CP-SAT solver [29]. We conducted our experiments using a Python implementation on a machine with 32 GB RAM and an 8-core 2.1 GHz CPU.

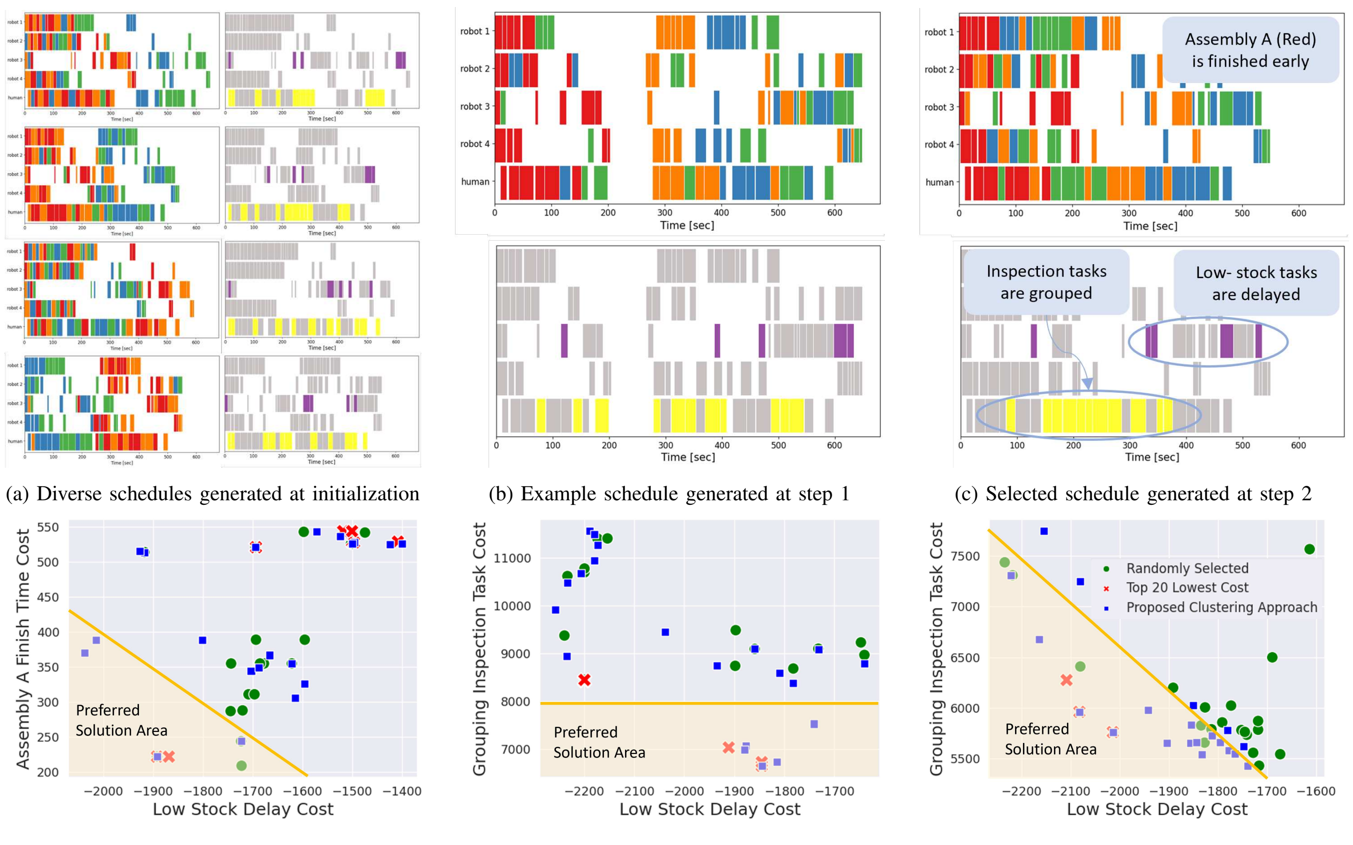
B. Characterization of Diverse Plan Generation

In practice, when generating diverse plans, we do not aim to find the optimal solution in each iteration, but instead, we approximate the solution and set a computation time limit (of 10 seconds for the following experiments) for each iteration after the initialization step. We evaluate our diverse plan generation approach, where we vary how large and constrained the problem is, and examine the number of solutions found, the average makespan, and the diversity of the population of solutions. To evaluate diversity, we calculate the L1 distance for all pairs in the solution populations, which is the sum of the absolute difference in tasks' start times in a pair of schedules. We calculate the average L1 distance for all pairs in the solution population and normalize it with the increase factor for task size ($n \cdot 50$) and the optimal makespan (t_{opt}) : $L_{1,Norm} = L1/(n * t_{opt})$. Lastly, we set a bound on a feasible solution's makespan to be within 20% of the optimal makespan.

We found that the CP-SAT solver finds the optimal solution to the problem for all 50, 100, and 150 tasks with multiples of three different sequential constraint counts 50 (low), 114 (medium), and 314 (high) within 10 seconds. After initialization, at iteration 1, we see a small number of explored solutions which is due to the solver effectively pruning the search space at the first iteration. However, with an increase in the number of iterations, the solver consistently finds a large number of solutions that are within 20% of the optimal solution. For almost all problem inputs, we also see an increase in both the number of solutions explored and the associated diversity score, indicating that our approach can generate diverse solutions effectively. An interesting observation is that the diversity scores in our planning process have a higher standard deviation. We reason that this is due to intermediate solutions from each iteration being situated between the more diverse solutions, contributing to a broader spread of diversity scores. Despite being an incremental planning approach where we repeatedly called the CP-SAT solver, we found that at least 95% of all aggregated solutions were unique, indicating the approach finds different, diverse solutions efficiently and there is no wasted search. Additionally, we observe higher L1 distances for problems with lower numbers of sequential constraints, and we reason that having more parallel constraints leads to higher chances of generating more diverse solutions.

C. Case Study

We have developed a case study where a team comprising 4 robots and 1 human is tasked with completing four different assemblies, denoted by different colors (red, green, orange, and blue). In this case study, we introduce two important terms: human's "latent preferences", which are the preferences that the human operator has but are unknown to the system, and "specified preferences", which are the preferences that the human operator has become aware of



(a) Diverse schedules generated at initialization (b) Example schedule generated at step 1 (c) Selected schedule generated at step 2

(d) Selected plans for two latent preferences (e) Selected plans for specified & latent preference (f) Selected plans for two specified preferences

Fig. 4: We show representative schedules generated during the preference elicitation and incorporation process. We also visualize the selected solutions for different latent preference spaces. We compare the selected solutions for our proposed plan selection approach and 2 baselines.

through the preference elicitation process. Our goal in this case study is to elicit the human's latent preferences as well as determine a preferred schedule that implicitly balances these preferences.

We have defined the following latent preferences for the humans: (P1) the human prefers Assembly A (red) to be finished as soon as possible, (P2) they want to delay tasks that require a part that is low in stock (purple) and schedule them for later when more parts will be delivered, and (P3) they need to do inspection tasks (yellow), and they prefer to group the tasks so that they can perform them all at once. The preference soft constraint and penalty are defined in VI.

We simulated this case study in three steps and selected schedules that satisfied the case study story. The schedules in this case study showed that the preferred solution has the following range of cost for each preference: [$P1$: (0)-(425), $P2$: (-2300) - (-1700), $P3$: (5000)-(8000)]. We denote this range as the preferred solution area in Figure 4 . We now briefly describe the case study. We show generated schedules and show where these solutions would lie in the latent preference space if they were known as apriori. For each step, we select 20 schedules to present to the human.

Initialization step: The system generates diverse solutions that minimize assembly makespan. The human specifies the following task features of interest: assembly type and whether tasks involve low-stock parts or inspection. Then, the system generates solution embeddings, clusters the solutions, and selects 20 schedules to present to the human. A subset of the generated schedules is shown in Figure 4a, and we see that they are diverse with respect to the task features of interest. The human notices that assembly A tasks (red) can be prioritized and low-stock part tasks can be delayed so preferences $P1$ (assembly A) and $P2$ (low stock) are specified. This can be visually confirmed from Figure 4a, and we can see plans presented by the system (blue squares) lie within the latent preferred solution area.

Step 1: The system has generated another set of diverse solutions, and a representative solution is shown in Figure 4b. As we can see, assembly A's makespan is minimized (this is the case for all solutions), and the low-stock part tasks are delayed as much as possible, resulting in an increase in makespan up to the specified 20% suboptimal bound. The user notices that the inspection tasks (yellow) are spread out and would prefer to group them together. Therefore, they specify preference $P3$ (inspection). In Figure 4e, we compare the latent preference $P3$ with the previously specified preference $P2$ (low stock). The cost range for $P2$ has significantly reduced because the system would have tried to minimize the cost. For $P3$, we see significant coverage in solutions, indicating that humans would be informed of plans in the latent solution area where their preference is either strongly satisfied or strongly violated.

Step 2: In step 2 of the system, a new set of solutions is generated that minimizes all specified preferences in addition to makespan. Figure 4f shows that preferences $P2$ and $P3$ are conflicting objectives, and the system has approximated a pareto solution for those two preferences effectively. The selected schedule is shown in Figure 4c. We can see that all preferences have been met, and the chosen solution has given priority to the inspection grouping preference over the low-stock task preference.

D. Characterization of Preference Elicitation

We also briefly discuss the proposed preference elicitation strategy in the context of the case study. In addition to our solution clustering approach, we have compared it with two other methods: 1) selecting the top 20 solutions based on the predetermined cost (red cross) and 2) randomly selecting solutions from a diverse solution population (green circle) which is seen in Figures 4d, 4e, 4f. Here we show that by only specifying task features of interest and intelligently clustering the solutions, we were able to generate diverse solutions that provided strong coverage of the latent preference space much better than randomly selecting diverse plans or presenting the best pre-defined low-cost plans.

VIII. Conclusion

We have developed a framework that incorporates human preferences into task scheduling. Our approach involves an iterative process where we generate high-quality and diverse plans, then elicit preferences by intelligently selecting and presenting a representative subset of plans based on task features of interest, and finally incorporate these preferences as soft constraints using a large-language model. Our results show that our diverse plan generation method can efficiently generate schedules that are both of high quality and diverse. Further evaluation of our framework on an assembly-inspired case study with multiple complex preferences shows that the system can effectively select diverse schedules to elicit multiple human preferences, incorporate them, and converge to a preferred schedule that balances multiple preferences.

Acknowledgements

This work was supported by the National Science Foundation NRI (# 2024936).

References

[1] H. Nemlekar, N. Dhanaraj, A. Guan, S. K. Gupta, and S. Nikolaidis, “Transfer learning of human preferences for proactive robot assistance in assembly tasks,” in *Proceedings of the 2023 ACM/IEEE International Conference on Human-Robot Interaction*, 2023, pp. 575–583.

[2] M. C. Gombolay, C. Huang, and J. Shah, “Coordination of human-robot teaming with human task preferences,” in *2015 AAAI Fall Symposium Series*, 2015.

[3] A. Jorge, S. A. McIlraith *et al.*, “Planning with preferences,” *AI Magazine*, vol. 29, no. 4, pp. 25–25, 2008.

[4] E. C. Grigore, A. Roncone, O. Mangin, and B. Scassellati, “Preference-based assistance prediction for human-robot collaboration tasks,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4441–4448.

[5] H. Nemlekar, J. Modi, S. K. Gupta, and S. Nikolaidis, “Two-stage clustering of human preferences for action prediction in assembly tasks,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 3487–3494.

[6] D. Smith, “Planning as an iterative process,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 26, no. 1, 2012, pp. 2180–2185.

[7] J. Kim, C. Banks, and J. Shah, “Collaborative planning with encoding of users’ high-level strategies,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[8] M. Li, Z. She, A. Turrini, and L. Zhang, “Preference planning for markov decision processes,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, 2015.

[9] M. Li, A. Turrini, E. M. Hahn, Z. She, and L. Zhang, “Probabilistic preference planning problem for markov decision processes,” *IEEE transactions on software engineering*, vol. 48, no. 5, pp. 1545–1559, 2020.

[10] T. C. Son and E. Pontelli, “Planning with preferences using logic programming,” *Theory and Practice of Logic Programming*, vol. 6, no. 5, pp. 559–607, 2006.

[11] A. Gerevini and D. Long, “Plan constraints and preferences in pddl3 the language of the fifth international planning competition,” *ICAPS 2006*, 01 2005.

[12] S. Baier and S. A. McIlraith, “Htn planning with preferences,” in *21st Int. Joint Conf. on Artificial Intelligence*, 2009, pp. 1790–1797.

[13] D. Braziunas and C. Boutilier, “Elicitation of factored utilities,” *AI Magazine*, vol. 29, no. 4, pp. 79–79, 2008.

[14] M. Katz, S. Sohrabi, O. Udrea, and D. Winterer, “A novel iterative approach to top-k planning,” in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 28, 2018, pp. 132–140.

[15] S. Mantik, M. Li, and J. Porteous, “A preference elicitation framework for automated planning,” *Expert Systems with Applications*, vol. 208, p. 118014, 2022.

[16] M. Das, P. Odom, M. R. Islam, J. R. J. Doppa, D. Roth, and S. Natarajan, “Planning with actively eliciting preferences,” *Knowledge-Based Systems*, vol. 165, pp. 219–227, 2019.

[17] M. Fox, D. Long, and D. Magazzeni, “Explainable planning,” *arXiv preprint arXiv:1709.10256*, 2017.

[18] B. Krarup, M. Cashmore, D. Magazzeni, and T. Miller, “Model-based contrastive explanations for explainable planning,” 2019.

[19] M. Gombolay, A. Bair, C. Huang, and J. Shah, “Computational design of mixed-initiative human–robot teaming that considers human factors: situational awareness, workload, and workflow preferences,” *The International journal of robotics research*, vol. 36, no. 5-7, pp. 597–617, 2017.

[20] P. Abbeel and A. Y. Ng, “Apprenticeship learning via inverse reinforcement learning,” in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 1.

[21] W. Wang, R. Li, Y. Chen, Z. M. Diekel, and Y. Jia, “Facilitating human–robot collaborative tasks by teaching-learning-collaboration from human demonstrations,” *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 2, pp. 640–653, 2018.

[22] N. Dhanaraj, S. V. Narayan, S. Nikolaidis, and S. K. Gupta, “Contingency-aware task assignment and scheduling for human-robot teams,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 5765–5771.

[23] L. Ingmar, M. G. de la Banda, P. J. Stuckey, and G. Tack, “Modelling diversity of solutions,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 02, 2020, pp. 1528–1535.

[24] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, “graph2vec: Learning distributed representations of graphs,” *arXiv preprint arXiv:1707.05005*, 2017.

[25] J. H. Kang, N. Dhanaraj, W. Siddhant, and G. Satyandra K., “Reactive failure recovery and scheduling for human-robot teams,” in *2024 IEEE International Conference on Robotics and Automation (ICRA)*.

[26] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, “Progprompt: Generating situated robot task plans using large language models,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[27] N. Dhanaraj, N. Ganesh, R. Gurav, M. Jeon, O. M. Manyar, S. Narayan, J. Park, Z. Yu, and S. K. Gupta, “A human robot collaboration framework for assembly tasks in high mix manufacturing applications,” in *International Manufacturing Science and Engineering Conference*, vol. 87240. American Society of Mechanical Engineers, 2023, p. V002T07A011.

[28] J. H. Kang, N. Dhanaraj, O. Manyar, S. Wadaskar, and S. K. Gupta, “A Task Allocation and Scheduling Framework to Facilitate Efficient Human-Robot Collaboration in High-Mix Assembly Applications,” in *Proceedings of the ASME Manufacturing Science and Engineering Conference*, Knoxville, TN, June 2024, to appear.

[29] L. Perron and V. Furnon, “Or-tools,” Google. [Online]. Available: https://developers.google.com/optimization/