Human-Centered ASP Applications: Representation & Reasoning

Aysu Boğatarkan
Sabancı University, Computer Science and Engineering, İstanbul, Turkey

Müge Fidan
Sabancı University, Computer Science and Engineering, İstanbul, Turkey

Esra Erdem[1]
Sabancı University, Computer Science and Engineering, İstanbul, Turkey

Abstract

As the objective of Artificial Intelligence (AI) changes towards building rational agents that are provably beneficial for humans, Knowledge Representation and Reasoning (KRR) plays an important role in addressing the user-oriented challenges in applications, such as generality, flexibility, provability, hybridity, bi-directional interactions, robustness, and explainability. In this tutorial, we will introduce participants to modeling and solving problems in human-centered real-world applications, using KRR methods and tools of Answer Set Programming (ASP), while addressing such challenges for AI.

2012 ACM Subject Classification Computing methodologies → Knowledge representation and reasoning

Keywords and phrases Answer set programming, human-centered applications, multi robot planning in warehouses, explainability, stable roommates problem, usefulness evaluations

Digital Object Identifier 10.4230/OASIcs.RW.2024/2025.4

Category Invited Paper

1 Introduction

Answer Set Programming (ASP) [11, 13, 10], based on answer set semantics for logic programs [6, 7], is a knowledge representation and reasoning paradigm with expressive languages and efficient solvers. It provides a formal framework for elaboration tolerant representations [12] and a wide range of reasoning tasks, e.g., nonmonotonic reasoning and abductive reasoning. Hence, ASP has been used in many applications, e.g., that involve declaratively solving combinatorial search problems, query answering over large knowledge bases, and reasoning about actions and change in a dynamic world.

As the objective of Artificial Intelligence (AI) changes towards building rational agents that are provably beneficial for humans, real-world applications necessitate methods that are general, flexible, robust, explainable, hybrid, and interactive. Since ASP provides a formal framework for elaboration tolerance in representations and variety in reasoning, it provides an appropriate platform to develop such methods.

In this tutorial, we will study two real-world examples, to illustrate the use of ASP for such human-centered applications. In one of these examples, the goal is to build an AI system that can plan the actions of multiple robots that pick and deliver items in a warehouse, and provide answers and explanations to the end-user engineers with their queries about these plans. In the other example, the goal is to build an AI system that can assign roommates to students at a college, considering various constraints and preferences to benefit the end-users (e.g., students, dormitory staff, college administration).

[1] Corresponding author

2 Planning the Actions of Robots at a Warehouse

Consider an automated warehouse with multiple robots working in it. The task of the robots is to carry items from various parts of the warehouse to other parts, for instance from shelves to packing stations. Since the warehouse is a shared environment for the robots, they need to avoid collisions among each other, as well as with obstacles blocking their way, while fulfilling their tasks. Furthermore, the tasks of the robots should be completed within a time limit, e.g., by the end of the day, for the warehouse to operate successfully. This problem is called the *Multi-Robot Planning (MRP) problem*. A solution to MRP is a continuous plan, i.e., continuous trajectories for robots to follow starting from their initial configurations until their goal configurations. Theoretically, such a continuous plan can be computed using motion planning algorithms. However, in practice, this is not possible due to high dimensions of the configuration spaces.

2.1 Simplifying MRP as a Discrete Problem

Instead of finding continuous plans for the agents, it is possible to simplify this problem by relaxing it into a discrete problem.

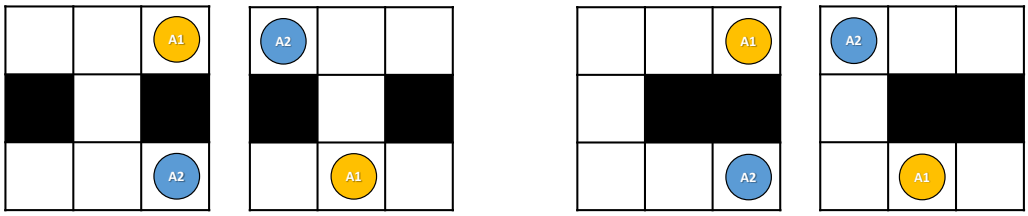
One relaxation that can be done is considering the time as discrete time steps instead of continuous time, and the limit on the time as a limit on maximum or total number of time steps of the resulting plans.

Similarly, instead of considering the whole warehouse as a continuous space, we can represent it as a graph. We can divide the environment into smaller areas and represent each area as a vertex and add undirected edges between every two adjacent small area. Furthermore, we can choose these smaller areas in such a way that each robot can fit inside a vertex and each vertex can contain at most one robot. In this way, we do not need to consider the size or shape of the robots while computing our plans.

With these relaxations, we do not have to plan for every instant of time and we do not need to consider every point of the environment while planning. We will only consider at which vertex the robots are located at, for every time step. At each step, the robots can either stay at their current locations or move to an adjacent vertex. From the positions specified as the initial and goal configurations of the robots, we can identify at which vertex the robot starts at and at which vertex it should arrive at the end as well.

This representation of the environment and the the idea of having a solution for discrete time steps is also useful when we think about possible collisions. For the collisions between the robots, since only one robot can fit inside a vertex due to how we constructed our graph, we can say that no two robots can be at a vertex at the same time. Similarly, if we think about the edges of our graph, we can say that at most one robot can fit on an edge (due to how we set our vertices), and we can identify the collisions occurring on the edges when they are shared by two agents at the same time interval. Remember that with our simplification, we only need to compute which vertex the robot visits at each time step, so we would not have explicit information about the time at which an edge is visited by a robot. Thus we need to consider the vertices at the endpoints of the edges and at which step they are visited. Note that the robots are not allowed to wait on an edge with our simplification, so the endpoints of an edge would be visited in consecutive time steps.

This discretized problem is called *Multi-Agent Path Finding (MAPF) problem* [15]. It is a combinatorial search problem with different application areas, including automated warehouses [16].



(a) (b) (c) (d)

Figure 1 Two MAPF instances, with initial and goal configurations (a) & (b) for Instance 1 and (c) & (d) for Instance 2, respectively.

2.2 Mathematical Modelling of MRP as MAPF

First, let us precisely describe the input of the simplified MRP problem, according to our discussions above:

- a set $A = \{a_1, a_2, \ldots, a_n\}$ of agents representing the robots,
- an undirected graph $G = (V, E)$ representing the environment,
- an integer τ, denoting the upper bound on the time steps,
- a function $init : A \to V$ that maps agents to their initial locations (vertices),
- a function $goal : A \to V$ that maps agents to their goal locations,
- a set $O \subseteq V$, denoting the vertices blocked by obstacles.

Given these input, the aim is to find the collision-free plans of the robots (i.e., the mapping from time steps to vertices from initial vertex to goal vertex). Then the output consists of the following, for every agent $a_i \in A$, for some positive integer $u \leq \tau$:

- a path $P_i = \langle v_{i,0}, \ldots, v_{i,l_i} \rangle$ of finite length l_i ($l_i \leq u$) such that
  - $v_{i,j} \in V \setminus O$ (i.e. no collisions with obstacles),
  - for every $\langle v_{i,j}, v_{i,j+1} \rangle$ in P, there exists an edge $\{v_{i,j}, v_{i,j+1}\} \in E$ (i.e. the path is connected)
  - $v_{i,0} = init(a_i)$ (i.e. the path starts from the agent's initial location),
  - $v_{i,l_i} = goal(a_i)$ (i.e. the path ends at the agent's goal location), and
- a traversal f_i of the path P_i, where f_i is a function that maps every integer t ($0 \leq t \leq u$) to a vertex in P_i such that
  - for every t, if $f_i(t) = v_{i,k}$ then $f_i(t+1) = v_{i,k}$ or $f_i(t+1) = v_{i,k+1}$, and
  - for every other agent $a_j \in A$ with traversal f_j of path P_j,
    - for every t where $0 \leq t \leq u$, $f_i(t) \neq f_j(t)$ (i.e no two agents are at the same vertex at the same time), and
    - for every t where $0 \leq t < u$, if $f_i(t) = f_j(t+1)$, then $f_i(t+1) \neq f_j(t)$ (i.e no two agents can swap their locations).

▶ **Exercise 1.** *Consider the two MAPF instances with two agents A1 and A2, shown in Fig. 1. In these figures, (a) and (b) show the initial configuration and the goal configuration of agents in Instance 1, respectively, while (c) and (d) show the initial configuration and the goal configuration of agents in Instance 2, respectively. For both instances, the black cells denote the obstacles. Can you find a solution to each one of these MAPF instances in $\tau = 4, 5, 6, \ldots$ time steps?*

2.3 Representing MAPF in ASP

Now that we have a mathematical model of MAPF problem, let us represent it formally in ASP.

Signature

Let us use the following atoms to describe the input of our mathematical model:

$time(t)$:	t is a time step from 0 to the upper bound T
$agent(a)$:	a is an agent in A
$vertex(x)$:	x is a vertex in V
$edge(x, y)$:	$\langle x, y\rangle$ is an edge in E
$init(a, x)$:	x is the initial location of agent a
$goal(a, x)$:	x is the goal location of agent a
$obstacle(x)$:	vertex x is occupied by an obstacle

For the output, we will compute the plans of the agents, denoting where the agent is at which step. For this purpose, we introduce the atom $plan(a, t, x)$, meaning that agent a is located at vertex x at time t.

Plan generation

We can compute our plans recursively starting from time 0 and the initial locations of the agents. At each time step, agents can wait at their current locations or move to an adjacent vertex.

$$plan(a, 0, x) \leftarrow init(a, x) \quad (a \in A).$$
$$1\{plan(a, t, x);\ plan(a, t, y) :\ edge(x, y)\}1 \leftarrow plan(a, t{-}1, x) \quad (a \in A, 0{<}t{\leq}T).$$

Goal constraints

There should be an incoming edge to the goal locations of agents, to make sure that every agent reaches its goal:

$$\leftarrow \{plan(a, t, y) : edge(x, y), goal(a, y)\}0 \quad (a \in A, 0{\leq}t{\leq}T).$$

Furthermore, to ensure that the plan of a new agent a ends at the goal, i.e., there is no outgoing edge from the goal, we add the following constraints:

$$\leftarrow plan(a, t, x), plan(a, t{+}1, y), edge(x, y), goal(a, x) \quad (a \in A, 0{\leq}t{<}T).$$

No collisions

Now that we have generated plans for all agents, we need to ensure that there are no collisions.

To avoid vertex collisions, we ensure that no two agents a_1 and a_2 are at the same vertex x at the same time.

$$\leftarrow plan(a_1, t, x), plan(a_2, t, x) \quad (a_1{\neq}a_2, 0{\leq}t{\leq}T, v \in V).$$

To avoid edge collisions, we ensure that no two agents a_1 and a_2 swap their locations at the same time.

$$\leftarrow plan(a_1, t, x), plan(a_2, t, y), plan(a_1, t{-}1, y), plan(a_2, t{-}1, x)$$
$$(a_1{\neq}a_2, 0{<}t{<}T, \{x, y\} \in E).$$

To avoid obstacle collisions, we ensure that no agent a visits a vertex x containing an obstacle:

$$\leftarrow plan(a,t,x), obstacle(x) \quad (0{\leq}t{\leq}T).$$

▶ **Exercise 2.** *Verify your answers to the question in Exercise 1, using the ASP program above with an ASP solver.*

2.4 Generalizing MAPF for Real-World Applications: Flexible Solutions

Although our model of MAPF addresses the main problem of planning routes for multiple agents in a shared environment, solving MAPF may not be sufficient for addressing other challenges of a real-world automated warehouse.

For instance, since the robots in an automated warehouse are mobile robots, they are equipped with batteries. The battery levels of robots will decrease as they move around and therefore they may need charging. To address this, we can extend our mathematical model to take battery levels into account, as well as charging stations in the environment. Integrating battery levels and charging into our model and encoding would benefit the users of our methods, since they would not have to keep track of when the robots need charging.

Another challenge that is not addressed by MAPF occurs when we consider a warehouse where robots and humans work together in a shared environment. To ensure the safety of human workers, robots may need to move slower than usual in the specific parts of the warehouse. If we extend our model to consider different speeds for the robots, we can generate plans that would be more useful in this setting, by ensuring the safety of human workers.

(a)

Time	A1 Location	A2 Location
0	1	12
1	5	8
2	9	4
3	10	transit
4	11	3
5	12	transit
6	8	2
7	12	transit
8	-	1

(b)

Time	A1 Location	A2 Location
0	1	12
1	transit	8
2	2	4
3	transit	8
4	3	12
5	transit	11
6	4	10
7	8	9
8	12	5
9	-	1

(c)

Figure 2 (a) A1 and A2 denote the initial positions of Robots 1 and 2; each robot aims to reach the diagonally-opposite corner. Cell 8 is a waypoint for Robot 1 and Cell 4 is a waypoint for Robot 2. Yellow cell is the charging station, red cells are the slow zone, and black cells are obstacles. (b) is an optimal plan for this instance. (c) is an alternative plan with a longer plan length.

As an example, consider the instance shown in Figure 2(a) in a small warehouse, viewed as a 3x4 grid. The warehouse contains a shelf unit that occupies two grid cells, denoted as obstacles shown by the black cells. As the robots move, their battery level decreases by 1 at each time step and they may need charging. For this purpose, the warehouse contains a charging station, denoted by the yellow cell located at Cell 10. If a robot is at a charging station, its battery level may quickly get to the maximum level or the robot can move forward without charging its battery. In the corridor denoted by red cells, the robots should move slowly, due to humans working nearby. Normally, it takes one time step for a robot to move from one grid cell to the other, but in this slow corridor it takes two time steps to move from one cell to the next. When the robot is located between two grid cells, we say that it is "*in transit*". There are two robots, $A1$ and $A2$, in this warehouse and they are initially

located in two corners, denoted by the colored circles. The robots aim to swap their places at the end. The stars denote the waypoint of the same colored robot and the robot must visit its waypoint to pick up items on its way to its goal location. An optimal solution for this instance is illustrated in Figure 2(b).

▶ **Exercise 3.** *How can we update the ASP program described in the previous scetion for MAPF, to be able to solve this problem?*

While working on Exercise 3, we need to keep in mind that the end-users may request further extensions, so elaboration tolerance is important for a general and flexible representation [2].

2.5 Interactions with the End-Users: Explainability

In addition to the extensions to the problem to address more challenges, the engineers who will execute the solutions computed by our methods may want to learn more about the solution that they want to execute in the warehouse.

For instance, the engineer may ask about whether some modifications on the executed plan would be feasible or not. A modification can include changing a location visited by an agent or changing when an agent waits at a location. Moreover, if we consider the extensions discussed above, the engineer may want to check if it is feasible not to charge a specific robot.

Providing only a yes/no answer about feasibility may not be very informative for the users. In addition to an answer about feasibility, we can provide explanations on why this modification is feasible or not.

Explaining infeasibility and counterfactuals

Suppose that a modification is found infeasible, then, it is desirable to provide an explanation regarding the infeasibility of the modification. An explanation about infeasibility could be "due to collisions with obstacles or other robots", or "due to low battery-level.".

For example, consider the instance in Figure 2(a) and an optimal solution for this instance, Figure 2(b). Now assume that the engineer wants to modify the solution, such that Robot 1 does not visit Cell 11 and it is found infeasible. Then the engineer asks the following question:

> "Why does Robot 1 visit Cell 11 (at any time, in an optimal solution)?"

We would like to generate explanations by answering the question: "What will happen if Robot 1 does not visit Cell 11?" and generates the following explanation:

> "Robot 1 has to visit at Cell 11 in an optimal plan; otherwise, Robot 2 would not be able to visit its waypoint at Cell 4."

Confirming feasibility and suggesting alternatives

Suppose that the modified solution is found feasible. Then, in addition to confirming the feasibility of the plan, it would be useful to provide the alternative solutions to the engineer.

We can also think about the optimality of solutions when providing explanations. If the modification requested by the user is found feasible, we can further check whether the modified solution is optimal or not.

Explaining nonoptimality and suggesting suboptimal solutions

Suppose that the modification is found feasible only when the modified solution is suboptimal. An explanation regarding nonoptimality of the modified solution could be "because some more time is needed to complete tasks" or "because some more charging is required".

Consider the example above, and a modified solution where Robot 1 does not visit Cell 11. In addition to the explanation above, it is also possible to generate an alternative suboptimal plan (Plan 2 described in Figure 2(c)) with the following explanation:

> "Robot 1 has to visit Cell 11, otherwise the plan will not be optimal. Here is an alternative plan with a longer plan length: Plan 2..."

Explaining nonoptimality and counterfactuals

Consider the same instance and assume that the initial the battery level of Robot 1 is 5 and the initial battery level of Robot 2 is 10. Plan 1 in Figure 2(b) is also an optimal solution for this instance. In the optimal solution, Robot 1's battery is fully charged at time step 3 when it visits the charging station at Cell 10, so its battery level increases to 10 at time step 4.

Suppose that an engineer asks the following query about Plan 1:

> "Why does Robot 1 charge at Cell 10 (at any time)?"

To answer this question, we can check whether there is an optimal plan where Robot 1 does not have to charge at Cell 10. However, there are no other solutions with where Robot 1 does not charge at Cell 10. Then, we can try to generate explanations by answering the question instead:

> "What would happen if Robot 1 does not charge Cell 10 in the current plan?"

This counterfactual question generates the following explanation:

> "Robot 1 has to charge at Cell 10 in the current plan; otherwise, it will run out of battery at time step 5."

We study explainability for MAPF in our earlier work [1], considering feasibility and optimality of solutions, nonexistence of solutions and observations about solutions, utilizing counterfactuals and violations to generate explanations and suggestions.

3 Assigning Roommates for Students at a Dormitory

Imagine a group of students applying for accommodation at a college dormitory, and thus need to be assigned roommates. At the time of dormitory applications, each student is requested to provide their roommate preferences over other students. The challenge is to find a roommate assignment that respects these preferences as much as possible, while minimizing the number of disappointed students. Under some conditions, this problem boils down to the *Stable Roommates problem (SR)*, well-studied problem in economics and game theory [5].

More precisely, SR is defined for an even number of agents, and characterized by the roommate preferences of each agent over other agents; here, it is assumed that each agent ranks all others in strict order of preference and that the preferences are characterized by lists. A solution to SR is then a matching (i.e., a partition of the agents into pairs) so that the following two conditions hold. First, the matched agents are *acceptable* to each other

(i.e., they are in the preference lists of each other). Next, no student is disappointed with the matching, i.e., there is no pair of students who prefer each other to their current roommates and may "block" the matching. A matching that avoids such *blocking pairs* is called *stable*.

However, the scenarios in the real world are rarely so neat. For example, students may not know everyone, and they may be indifferent between some of them. This observation has lead to an extensive study of variations of SR, e.g., with incomplete preference lists (SRI) [8], with preference lists including ties (SRT) [14], and with incomplete preference lists including ties (SRTI) [9].

Furthermore, as noted by Gale and Shapley [5], there is no guarantee in finding a stable matching to every SR problem instance; but, in real world, the students should be assigned a roommate somehow. This observation has led to optimization variants of SR, like Almost SRTI, where the number of disappointed students is minimized.

In this tutorial, we will illustrate how SRTI can be generalized for real-world applications. As a starting point, let us first provide a definition of SRTI and how it can be solved using ASP, as introduced in our earlier studies [3].

3.1 Mathematical Modelling of SRTI

Let A be a finite set of agents. For every agent $x \in A$, let $A_x \subseteq A \setminus \{x\}$ be a set of agents that are acceptable to x as roommates. We assume that x prefers y as a roommate compared to being single.

Each agent has a preference lists $\prec_x$, which is a partial order over A_x where incomparability is transitive, that is, if a student cannot compare x with y, and also cannot compare y with z, then we treat it as though they also cannot compare x with z. This avoids creating inconsistencies in the preference list. We denote by $y \prec_x z$ that x prefers y to z. An agent x is indifferent between the agents y and z, denoted by $y \sim_x z$, if $y \not\prec_x z$ and $z \not\prec_x y$.

An SRTI instance $(A, \prec)$ is then characterized by the set of agents and the collection $\prec$ of preference lists of all agents in A.

A matching for a given SRTI instance $(A, \prec)$ is a function $M : A \mapsto A$ such that, for all $\{x, y\} \subseteq A \times A$ where $x \in A_y$ and $y \in A_x$, $M(x) = y$ iff $M(y) = x$. If agent x is mapped to itself, then we say that it is single.

A matching M is blocked by a pair $\{x, y\} \subseteq A \times A$ $(x \neq y)$ if

B1 both agents x and y are acceptable to each other,

B2 x is single with respect to M, or $y \prec_x M(x)$, and

B3 y is single with respect to M, or $x \prec_y M(y)$.

A matching for SRTI is called stable if it is not blocked by any pair of agents.

▶ **Exercise 4.** *Consider a set of students {Alice, Bob, Carlos, Dave}, and the two different SRTI instances defined for them as shown in Figure 3. Do these instances have stable matchings? Please explain your answer briefly.*

Alice:	Bob Dave
Bob:	Carlos Alice Dave
Carlos:	Bob Dave
Dave:	{Bob, Carlos} Alice

Alice:	Bob Carlos Dave
Bob:	Carlos Alice Dave
Carlos:	Alice Bob Dave
Dave:	{Bob, Carlos, Alice}

Figure 3 Two SRTI instances.

3.2 Representing SRTI in ASP

An SRTI instance $I = (A, \prec)$ can be described by a set of facts in ASP, with atoms of the forms $agent(x)$ ("x is an agent in A") and $prefer2(x, y, z)$ ("agent x prefers agent y to agent z, i.e., $y \prec_x z$ ").

We assume that, for every agent x, assigning a roommate to x is preferred to not assigning anyone to x (e.g., for the dormitories to accommodate more students). Therefore, as part of input, for every $y \in A_x$, we include facts of the form $prefer2(x, y, x)$.

A matching $M : A \mapsto A$ for an SRTI instance is characterized by atoms of the form $room(x, y)$ ("agents x and y are roommates").

To represent SRTI problem in ASP, we also need to define some auxiliary concepts, like acceptability and blocking.

Acceptability

The mutual acceptability of agents is defined as follows:

$$accept(x, y) \leftarrow prefer(x, y, \_).$$
$$accept(x, y) \leftarrow prefer(x, \_, y).$$
$$accept2(x, y) \leftarrow accept(x, y), accept(y, x).$$

where $prefer$ is defined recursively as the transitive closure of $prefer2$:

$$prefer(x, y, z) \leftarrow prefer2(x, y, z).$$
$$prefer(x, y, z) \leftarrow prefer2(x, y, w), prefer(x, w, z).$$

Blocking pairs

Recall that a matching is described by atoms of the form $room(x, y)$. Then, considering the conditions B1–B3 above, a blocking pair (x, y) for a matching is defined by atoms of the form $block(x, y)$ where $x \neq y$:

$$block(x, y) \leftarrow accept2(x, y), single(x), single(y), not\ room(x, y).$$
$$block(x, y) \leftarrow accept2(x, y), single(x), like(y, x), not\ room(x, y).$$
$$block(x, y) \leftarrow accept2(x, y), like(x, y), single(y), not\ room(x, y).$$
$$block(x, y) \leftarrow accept2(x, y), like(x, y), like(y, x), not\ room(x, y).$$

Here, atoms of the form $single(x)$ describe single agents, i.e., the agents x who are not matched with a roommate:

$$single(x) \leftarrow room(x, x).$$

Atoms of the form $like(x, y)$ describe that agent x prefers agent y to its roommate $x' = M(x)$:

$$like(x, y) \leftarrow room(x, x'), prefer(x, y, x'). \quad (x' \neq y)$$

Generating stable matchings

For every agent x, exactly one mutual acceptable agent y is nondeterministically chosen as $M(x)$ by the choice rules:

$$1\{room(x, y){:}agent(y), accept2(x, y)\}1 \leftarrow agent(x).$$

The symmetry of this assignment is ensured by the following hard constraint:

$$\leftarrow room(x, y), not\ room(y, x).$$

Then, the stability of the generated matching is ensured by the hard constraints:

$$\leftarrow block(x, y) \qquad (x \neq y).$$

▶ **Exercise 5.** *Verify your answers to the question in Exercise 4, using the ASP program above with an ASP solver.*

3.3 Generalizing SRTI for Real-Word Applications: Knowledge-Based and Personalized Solutions

In real-world applications, when there is no stable matching to a given SRTI problem instance, we still need to find a "good enough" matching as the students should be assigned roommates. This observation has led to some reasonable relaxations of the problem, such as Almost SRTI. Here the goal is to minimize the number of blocking pairs, so a good enough matching is one with a small number of blocking pairs.

▶ **Exercise 6.** *How can we update the ASP program described for SRTI above, to be able solve Almost SRTI?*

However, there are still challenges when we consider the cases in the real world. For instance, most of the freshmen students often do not know any peers and leave their preferences empty. If most of the preference lists are empty, Almost SRTI may not lead to a good enough solution.

On the other hand, note that in many universities and colleges,[2] additional compatibility criteria, such as lifestyle habits or study environment preferences, are often collected through questionnaires. Can we utilize this additional information to find a "better" roommate matching, e.g., for such freshmen students? Let us try!

Additional personal information: appropriate and useful

First, we need to understand what kind of additional personal information can be requested from the users.

▶ **Exercise 7.** *For which following criteria, it would be appropriate and useful to request information from the students about themselves and about their future roommates: smoking habit, sleep habit, waking up habit, study habit, cleanliness, social habit, political view, religion, sexual preference, hobbies, favorite courses, favorite soccer teams, etc.?*

For that, we can consult the students by means of surveys, polls or interviews. For instance, in our earlier studies [4], we have constructed an online anonymous survey at Sabanci University and asked more than 150 students for their preferences over some plausible and useful criteria. Thanks to their responses, we have identified the most preferred five critera and their possible values:

[2] Cleveland Clinic Student Housing application: `https://my.clevelandclinic.org/-/scassets/files/org/professionals/student-housing/roommate-questionnaire-worksheet`

- smoking habits (smoker or nonsmoker?),
- cleanliness (clean, messy, indifferent?),
- room environment (quiet and study oriented, a social gathering place, a combination of both?),
- sleep habits (before 8am, 8am–10am, 10am–12pm, 12pm or later?), and
- study habits (in the room, outside the room, both?).

Representing additional personal information

Next, we need to formally describe this additional information as it will be a part of the input of the generalized SRTI problem, i.e., Personalized-SRTI.

Let B be a finite list $\langle b_1, b_2, \ldots, b_k \rangle$ of criteria. For each criterion $b_i \in B$, let C_i be a finite list $\langle c_{i1}, c_{i2}, \ldots, c_{im} \rangle$ of choices for b_i. For instance, consider the criteria list $B = \langle$"cleanliness", "sleep habits"$\rangle$. For each criterion, the choice lists can be defined as follows: $C_1 = \langle$"Clean", "Messy"$\rangle$ is the list of choices for "cleanliness", and $C_2 = \langle$"Goes to bed early", "Goes to bed before midnight", "Goes to bed after midnight"$\rangle$ is the list of choices for "sleep habits".

Let f be a function that maps an agent $x \in A$ and a criterion $b_i \in B$ to a positive integer j $(1 \leq j \leq |C_i|)$, describing the choice c_{ij} of the agent x. For instance, consider the example above and assume that Ayse is an agent in A. If Ayse's preference for the "cleanliness" criterion is "Clean", then f(Ayse,"cleanliness") $= 1$. If Ayse's preference for "sleep habits" criterion is "Goes to bed after midnight", then f(Ayse,"sleep habits") $= 3$.

For every agent $x \in A$, let us denote by $P_x = \langle f(x, b_1), f(x, b_2), \ldots, f(x, b_k) \rangle$ the choices of x for each criterion in B respectively. We refer to P_x as the agent x's *(preference) profile*.

Every criterion in B may have a different importance for each agent in A; this can be described by a weight function. Then, we can "sort" the profiles accordingly, considering the importance of criteria. For instance, the first element of Ayse's sorted profile includes (3,"sleep habits") if Ayse gives the most importance to "sleep habits", while agent Cem's sorted profile includes (5, "smoking") if Cem gives the most importance to "smoking habits".

For every agent $x \in A$, let us denote by P'_x the sorted profile of agent x, and by P' the collection of sorted profiles of all agents in A. Then a Personalized-SRTI instance, defined over a given SRTI instance $(A, \prec)$, is characterized by the triple $(A, \prec, P')$. How can we solve a Personalized-SRTI instance?

Since we already have a mathematical model for SRTI and an ASP-based method to solve it, one of the methods that we can investigate is to try to reduce Personalized-SRTI to SRTI. For that, we need to answer the following question: can we infer further preferences of agents from their sorted profiles?

Criteria-based personalized preference lists

Given sorted profiles of agents, first we generalize the concept of acceptability. Intuitively, two agents are "choice acceptable" to each other if they agree on a criterion (e.g., they are both nonsmokers).

Next, for every agent $x \in A$, we infer a new preference list $\prec'_x$. For two agents y and z in A that are choice acceptable to x, we say that x prefers y to z if the following holds:

- both y and z agree with x on all common criteria included in the first $j-1$ $(j > 0)$ elements of x's sorted profile, but
- y agrees with x on more number of common criteria included in the j'th element of x's sorted profile.

For example, consider three agents, Ayse, Buse and Duru, with the following sorted profiles:

$$\begin{aligned} P'_{Ayse} &= \langle\{(2, \text{“smoking”})\}, \{(1, \text{“cleanliness”})\}, \{(1, \text{“room environment”})\}, \\ &\quad \{(1, \text{“sleep habits”})\}, \{(1, \text{“study habits”})\}\rangle \\ P'_{Buse} &= \langle\{(3, \text{“study habits”})\}, \{(3, \text{“sleep habits”})\}, \\ &\quad \{(3, \text{“room environment”})\}, \{(2, \text{“smoking”})\}\rangle \\ P'_{Duru} &= \langle\{(2, \text{“smoking”})\}, \{(3, \text{“study habits”}), (3, \text{“sleep habits”}), \\ &\quad (3, \text{“room environment”}), (1, \text{“cleanliness”})\}\rangle \end{aligned}$$

Ayse and Buse are choice-acceptable to Duru. Consider the first element of P'_{Duru}, i.e., {(2,“smoking”)}. Both Ayse and Buse agree with Duru on smoking habits, so they are indifferent for Duru. Consider the next element of P'_{Duru}. Ayse agrees with Duru on cleanliness, while Buse agrees with Duru on study habits, sleep habits and room environment; since Buse agrees with Duru on more number of criteria, Buse $\prec'_{Duru}$ Ayse.

Solving Personalized-SRTI

Once a criteria-based personalized preference list $\prec'_x$ is inferred for each agent x, from the given sorted profiles P'_x of agents, we can append the inferred preference list $\prec'_x$ to the end of the preference list $\prec_x$ given by the agent. Let us denote these extended preference lists by $\prec''_x$. Then solving a Personalized-SRTI instance $(A, \prec, P')$ boils down to solving the SRTI problem $(A, \prec'')$.

With this approach, we expect obtaining longer preference lists for agents (e.g., for freshmen students), and increasing our chances to find a stable matching or a better matching. Is that really so in practice? Let us evaluate!

3.4 Usefulness Evaluations for Personalized-SRTI

Suppose that the originally given SRTI instance $(A, \prec)$ does not have a stable matching but the Personalized-SRTI instance $(A, \prec, P')$ built on top of it by augmenting additional personal information does have a stable matching. In this case, usefulness evaluations need to answer the following question:

> How good is a personalized stable matching, compared to unstable matchings for $(A, \prec)$?

Alternatively, suppose that both the originally given SRTI instance $(A, \prec)$ and the Personalized-SRTI instance $(A, \prec, P')$ have stable matchings. In this case, usefulness evaluations need to answer the following question:

> How good are the personalized stable matchings of $(A, \prec, P')$, compared to the stable matchings of $(A, \prec)$?

To be able to answer such questions, usefulness evaluations require feedback from a diverse set of end-users, via surveys, polls and interviews. However, designing and conducting human-subject empirical analysis is quiet challenging.

For instance, in our earlier studies [4], to answer the two questions above about the usefulness of personalized-SRTI, we have designed follow-up surveys with one or two short questions about a few small SRTI instances. These instances are described by text and figures for easier understandability, and the questions require short answers without increasing the cognitive load of the participants. Figure 4 illustrates one of the survey questions.

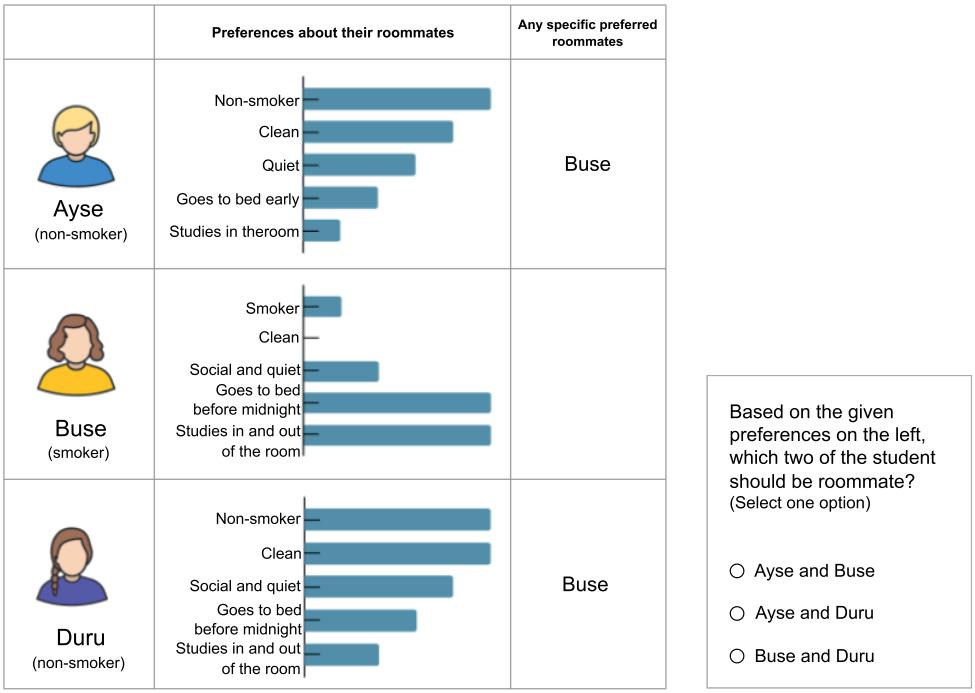


Figure 4 An example survey question: Given the preferences shown on the left hand side, choose the most reasonable roommates on the right hand side.

We have conducted these surveys with more than 200 students at Sabanci University as diverse as possible, after briefly informing them about the purpose of the study and getting their approvals. During these interactions, it is important not to misguide the participants and to introduce any biases.

The survey results confirm that extending SRTI to include further knowledge about the agents indeed provides practical benefits in finding stable/better matchings [4].

4 Conclusion

Human-centered real-world applications require a good understanding of the problems from the perspectives of different types of end-users, to be able to come up with a core mathematical model that captures the essence of the main problem, and that allows methods for efficient computation of solutions.

Then we can represent and solve the core problem using ASP. Thanks to the declarative feature of ASP, we can also provide guarantees about the correctness of the solutions.

However, solving the core problem is usually not sufficient for many real-world applications. We need to generalize the core problem and the proposed methods, considering the further needs of the end-users, allowing multi-modal interactions with them, and keeping in mind the computational and social aspects.

These more general methods requires evaluations not only from the computational point of view but also from the point of view of usefulness by the end-users. Designing and conducting human-subject empirical analysis via surveys, polls and interviews is quiet challenging, and deserves a separate tutorial.

It is important to emphasize that the human end-users are involved at each stage of a human-centered application, to help us understand the challenges of the real-world problems and to guide us design, develop and evaluate methods to solve these problems in such a way that the solutions benefit humans.

References

1 Aysu Bogatarkan and Esra Erdem. Explanation generation for multi-modal multi-agent path finding with optimal resource utilization using answer set programming. *Theory and Practice of Logic Programming*, 20(6):974–989, 2020. doi:10.1017/S1471068420000320.

2 Aysu Bogatarkan, Esra Erdem, Alexander Kleiner, and Volkan Patoglu. Multi-modal multi-agent path finding with optimal resource utilization. In *Proceedings of 5th International Conference on the Industry 4.0 Model for Advanced Manufacturing*, pages 313–324, 2020. doi:10.1007/978-3-030-46212-3_24.

3 Esra Erdem, Müge Fidan, David Manlove, and Patrick Prosser. A general framework for stable roommates problems using answer set programming. *Theory and Practice of Logic Programming*, 20(6):911–925, 2020. doi:10.1017/S1471068420000277.

4 Müge Fidan and Esra Erdem. Knowledge-based stable roommates problem: A real-world application. *Theory and Practice of Logic Programming*, 21(6):852–869, 2021. doi:10.1017/S1471068421000302.

5 David Gale and Lloyd Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

6 M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of ICLP*, pages 1070–1080, 1988.

7 Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991. doi:10.1007/BF03037169.

8 Dan Gusfield and Robert W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, Cambridge, MA, USA, 1989.

9 Robert W Irving and David F Manlove. The stable roommates problem with ties. *Journal of Algorithms*, 43(1):85–105, 2002. doi:10.1006/JAGM.2002.1219.

10 Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138:39–54, 2002. doi:10.1016/S0004-3702(02)00186-8.

11 Victor Marek and Mirosław Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer Verlag, 1999. doi:10.1007/978-3-642-60085-2_17.

12 John McCarthy. Elaboration tolerance. http://jmc.stanford.edu/articles/elaboration.html, 1999.

13 Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 1999. doi:10.1023/A:1018930122475.

14 Eytan Ronn. NP-complete stable matching problems. *Journal of Algorithms*, 11(2):285–304, 1990. doi:10.1016/0196-6774(90)90007-2.

15 Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Roman Barták, and Eli Boyarski. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proc. of SOCS*, pages 151–159, 2019. doi:10.1609/SOCS.V10I1.18510.

16 Peter R. Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29(1):9–20, 2008. doi:10.1609/aimag.v29i1.2082.